1970

A computer program for solution of sequence dependent routing problems using a branch-and-bound algorithm.

Jackson, Richard Alan

Monterey, California ; Naval Postgraduate School

A COMPUTER PROGRAM FOR SOLUTION OF SEQUENCE DEPENDENT ROUTING PROBLEMS USING A BRANCH-AND-BOUND ALGORITHM

by

Richard Alan Jackson

United States
Naval Postgraduate School

THESIS

A COMPUTER PROGRAM FOR SOLUTION OF
SEQUENCE DEPENDENT ROUTING PROBLEMS
USING A BRANCH-AND-BOUND ALGORITHM

by

Richard Alan Jackson

September 1970

A Computer Program For Solution of Sequence Dependent Routing Problems Using a Branch-And-Bound Algorithm

by

Richard Alan Jackson
Lieutenant, United States Navy
B.I.E., University of Florida, 1964

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1970

ABSTRACT

An algorithm for the solution of sequence-dependent routing problems is presented and programmed in FORTRAN IV for use on digital computers. Solutions, computation times and iteration requirements are summarized and discussed for eleven test cases.

With specific modification of the input data, a typical traveling salesman closed-loop problem may be solved by the same program.

TABLE OF CONTENTS

I.	INTRODUCTION	7
II.	THE ALGORITHM	9
III.	PROGRAMMING CONSIDERATIONS	17
IV.	THE COMPUTER PROGRAM	21
V.	TEST PROBLEMS	24
	A. SEQUENCE-DEPENDENT PROBLEMS	24
	B. TRAVELING SALESMAN PROBLEMS	26
VI.	COMPUTATIONAL RESULTS	32
VII.	CONCLUSIONS	35
APPENDIX A:	COMPLETE COMPUTER SOLUTION FOR TEST PROBLEM NO. TWO	36
APPENDIX B:	TYPICAL SOLUTION OUTPUT	37
APPENDIX C:	COMPOSITION OF COMPUTER CARD DECK	41
APPENDIX D:	DESCRIPTION OF VARIABLES USED IN COMPUTER PROGRAM	42
APPENDIX E:	COMPUTER PROGRAM	45
APPENDIX F:	MODIFICATIONS TO COMPUTER PROGRAM FOR DYNAMIC STORAGE ALLOCATION	60
BIBLIOGRAPHY		67
INITIAL DISTRIBUTION LIST		68
FORM DD 1473		69

4 (Blank)

LIST OF SYMBOLS AND ABBREVIATIONS

X - a subset of all feasible solution vectors

Y - a subset of X

$\bar{Y}$ - the complement of Y with respect to X

$W(X)$ - a bound on the objective function for all possible solution vectors in X

leg k - one of the sequence of arcs which form a complete route (the k-th leg of a route between N nodes is that arc (i, j) which is traversed between the k-th and $(k + 1)$-st nodes visited in sequence on the route)

arc(i, j) - a directed path from node i to node j

A_k, (a_{ij}^k) - the matrix of costs of traversing arc (i, j) on the k-th leg of the route

M_k, (m_{ij}^k) - the current working matrix of costs of traversing arc (i, j) on the k-th leg of route. (Initially $M_k = A_k$ but M_k is changed by the operations of the algorithm)

$g = \sum a_{ij}^k$ summed over the set of $(i, j{:}k)$ for committed arcs and legs

M_k' - the reduced form of M_k

$q(i_k, j_k{:}k)$ - the reducing constant for M_k

$\theta\,(i_p, j_p{:}k)$ - the second smallest element in M_k'

$\theta\,(i_o, j_o{:}k_o) = \max\limits_k \theta\,(i_p, j_p{:}k)$ where k is uncommitted

x - represents plus infinity as a matrix element

ACKNOWLEDGEMENT

It is with pleasure that I wish to acknowledge the staff and the operators of the W. R. Church Computer Facility at the U. S. Naval Postgraduate School for their gracious assistance and guidance during all stages of testing the computer program. The author is particularly indebted to William Erhman of the staff who provided the Assembly Language program included in Appendix F for dynamic allocation of storage space based upon the number of nodes in a given problem and the number of iterations desired.

I. INTRODUCTION

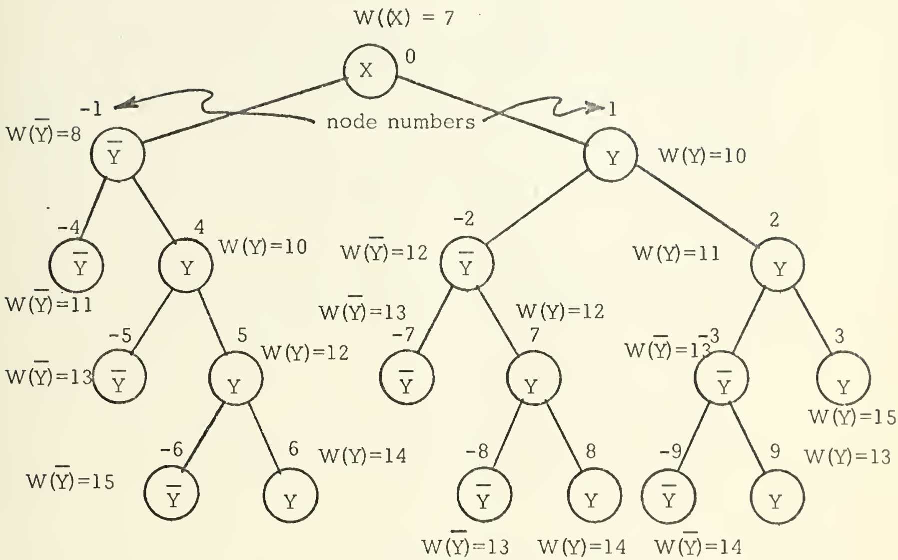
The algorithm programmed in this thesis, presented by DeHaemer [Ref. 1], uses the branch-and-bound technique to find the optimal route between N nodes. It determines the beginning and ending nodes and passes through each node exactly once. The criterion for optimality is to minimize total cost in traversing the (N-1) arcs of the route where the cost of traversing each arc is a_{ij}^{k}, which is a function of the k-th position in the sequence of arcs forming the route.

The purpose of this paper was to construct a computer program which would solve the general class of sequence-dependent routing problems using the above mentioned algorithm, given the matrices of all possible costs for each leg of the route. The difficulty in solving this class of problems has been in finding a method of selection of tours which avoids evaluation of all the (N-1)! possible tour costs in determining an optimal route.

Although several algorithms for typical traveling salesman problems have been proposed and programmed for a computer [Ref. 2], this paper presents the first program and results using the algorithm presented in the next section.

The operational results of solving several test problems are given along with a discussion of the limitations of the computer program. It is assumed that the reader is familiar with the branch-and-bound technique. References 1 and 3 discuss general background of branch-and-bound methods.

TYPICAL SEGMENT OF TREE



Notation:

W(X) - a lower bound on objective function for all possible solution vectors, attached to base node X of tree

Y - right-hand nodes with notation as follows (i,j:k) (i.e., k-th leg of route is from i to j)

W(Y) - lower bound associated with node Y

$\overline{Y}$ - left-hand nodes with notation as follows $(\overline{i,j}:k)$ (i.e., k-th leg of route is not from i to j)

$W(\overline{Y})$ - lower bound associated with node $\overline{Y}$

Note: For computer application, right-hand nodes are labeled with positive numbers and left-hand nodes are labeled with negative numbers.

Figure 1

II. THE ALGORITHM

The basic method employed by the algorithm is the branch-and-bound technique. The set of all possible routes through N nodes is broken up into smaller and smaller subsets and a lower bound on the cost of the best route in the subset is obtained. The bounds are then used as guides in determining further partitions into smaller subsets until the algorithm eventually isolates one or more subsets which are complete routes whose costs are less than or equal to the lower bounds for all other subsets. These routes are then declared optimal.

The algorithm generates a tree whose nodes represent subsets of routes as illustrated in Figure 1. The base node of the tree establishes an absolute lower bound on all possible routes. Each branch or segment of a branch is a complete route or subset of a complete route respectively. An example tree for an entire problem as generated by the computer program may be seen in Appendix A.

It is assumed that the set of matrices A_k can be specified for all (N-1) legs of the route. A problem with N nodes requires that (N-1) legs of a route be determined. Each leg k of a route is specified as being an arc (i,j) which is a directed path <u>from</u> node i <u>to</u> node j.

The algorithm as used for the computer program is listed here in complete detail. The first three test cases in Section V. A. are worked out in some detail in Ref. 1 and sufficient background of the algorithm may also be found in the same reference. The only modifications made

here in this algorithm are in the branching rule of step 8, elaboration of step 7 for the computer program, and in the branching to step 7 from step 4 when sufficient legs of a route are known so that a complete route may be specified.

<u>The Steps of the Algorithm</u>

Step 1:

The initial setup of the algorithm is made as follows:

1. Set $A_k = M_k$ for $k = 1,2,\ldots,(N-1)$.
2. X is the set of all possible routes.
3. Set $Z_o = \infty$ and Leg = 0. Z_o will be the cost of the optimal route at the end of the algorithm.

Step 2:

Find the minimum element in each matrix and reduce the matrices. An absolute lower bound on the cost of all tours is found.

1. For each leg k, $k = 1,2,\ldots,(N-1)$, find i_k, j_k, and $q(i_k,j_k:k)$ such that $q(i_k,j_k:k) = \min_i \min_j m^k_{ij}$.
2. Reduce M_k to M'_k where $m'^k_{ij} = m^k_{ij} - q(i_k,j_k:k)$ for all i,j, and k.
3. Label node X with $W(X) = \sum q(i_k,j_k:k)$ summed over $k = 1,2\ldots,(N-1)$. This label is the absolute lower bound on the cost of all tours.

Step 3:

Choose the subset for the next tree extension as follows:

1. $\theta(i_p, j_p : k) = \min_{ij \neq i_k j_k} m'^{k}_{ij}$ for each k where leg k is uncommitted.

2. $\theta(i_o, j_o : k_o) = \max_{k} \theta(i_p, j_p : k)$ where k ranges over the uncommitted legs.

3. Then $Y = (i_o, j_o : k_o)$ and $\overline{Y} = \overline{(i_o, j_o : k_o)}$ are the next branches from X.

Step 4:

Label $\overline{Y}$ by $W(\overline{Y}) = W(X) + \theta(i_o, j_o : k_o)$.

Step 5:[1]

Since an arc is to be committed to a leg, a new set of restricted matrices are formed by the following actions:

1. Delete M'_{k_o}.

2. a. Delete all elements in $M'_{k_o + 1}$ except row j_o.

   b. Delete columns i_o and j_o in $M'_{k_o + 1}$.

---

[1] Step 5 of the algorithm was accomplished in the computer program through the use of the variable matrix ARCCOM and the variable DEL which allowed only certain matrices and certain elements in these matrices to be considered in the succeeding steps.

3. a. Delete all elements in $M'_{k_o - 1}$ except column i_o.

   b. Delete rows i_o and j_o in $M'_{k_o -1}$.

4. Delete rows i_o and j_o and columns i_o and j_o in all M'_k except in $M'_{k_o + 1}$ and $M'_{k_o - 1}$.

5. Relabel the matrices as M_k.

6. Leg k is now committed to arc (i_o, j_o).

7. If (N-3) legs have been committed, go to Step 7.

Step 6:

Initiate procedures to determine what the next leg of the route should be.

1. For each k where leg k has not been committed to a route, find i_k, j_k, and $q(i_k, j_k{:}k)$ such that $q(i_k, j_k{:}k) = \min_i \min_j m^k_{ij}$.

2. Reduce M_k to M'_k for those legs k which are not committed and for all i, j of uncommitted arcs where $m'^k_{ij} = m^k_{ij} - q(i_k, j_k{:}k)$.

3. Label Y by $W(Y) = W(X) - \sum q(i_k, j_k{:}k)$ summed over k for uncommitted legs.

Step 7:[2]

Ascertain whether a route has been determined and if it has an upper bound which is equal to or less than Z_o.

1. Increment leg by one since a leg has been committed.
2. If (N-2) legs of route have been committed and $W(Y) \leq Z_o$, go to Step 10.
3. If (N-2) legs of route have been committed and $W(Y) > Z_o$, go to Step 8.
4. If (N-2) legs of route have not been committed and $W(Y) \leq Z_o$, go to Step 8, substep 4.
5. If (N-2) legs of route have not been committed and $W(Y) > Z_o$, go to Step 8.

Step 8:

Determine the node X from which to branch as follows:

1. Make the last Y node non-terminal since it is either the end of a complete route or the end of a segment of a complete route which has a cost which is greater than Z_o. Therefore, a search of $\bar{Y}$ nodes for suitable branch points must be made. Go to substep 2.

[2] Note that when (N-2) legs of route have been committed, the last leg is automatically determined and hence computation ends when (N-2) legs are known.

2. Choose the lowest numbered left-branch node with a label $W(\overline{Y}) \leq Z_o$ and branch from this node X. Go to Step 9. For all $\overline{Y}$ nodes with labels $W(\overline{Y}) > Z_o$, consider them non-terminal since they would all lead to higher cost routes. If there is no $\overline{Y}$ node which is a candidate for branching, go to substep 3.

3. All nodes have been made non-terminal by substeps 1 and 2 of this step and hence the optimal route has been found. STOP.

4. If substep three of Step 7 was satisfied, make last Y node to be the node X from which to branch. Make Y node non-terminal and set $W(X) = W(Y)$. Go to Step 3.

Step 9:

Set up the cost matrices and label node X as follows:

1. Set leg = 0. Then determine number of legs committed on limb of tree from which branch is to occur and set leg = to the number of Y nodes on the limb.

2. Compute $g = \sum a_{ij}^{k}$ summed over the set of $(i,j:k)$ for committed arcs and legs at this point in the tree.

3. If no legs have been committed, set $M_k = A_k$, otherwise ser $M_k^{'} = A_k$.

4. Carry out substeps 1 thru 4 of Step 5 for each of the committed arcs and legs.

5. Block paths which are not allowed (i.e., those left-hand nodes encountered on this branch of tree are forbidden nodes).

6. Carry out <u>Step 6</u> substeps 1 and 2.

7. Label X with $W(X) = g + \sum q(i_k, j_k:k)$ summed over k for the uncommitted legs.

8. Go to <u>Step 3</u>.

Step 10:

Determine complete route which has been found.

1. Arrange the committed arcs and legs to determine missing leg and arc on this leg.

2. Make last Y node non-terminal since a route was determined.

3. Set $Z_o = W(Y)$. Go to <u>Step 8</u>, substep 2.

<u>End of Algorithm</u>

A flow chart of the algorithm is in Figure 2.

FLOW CHART OF ALGORITHM

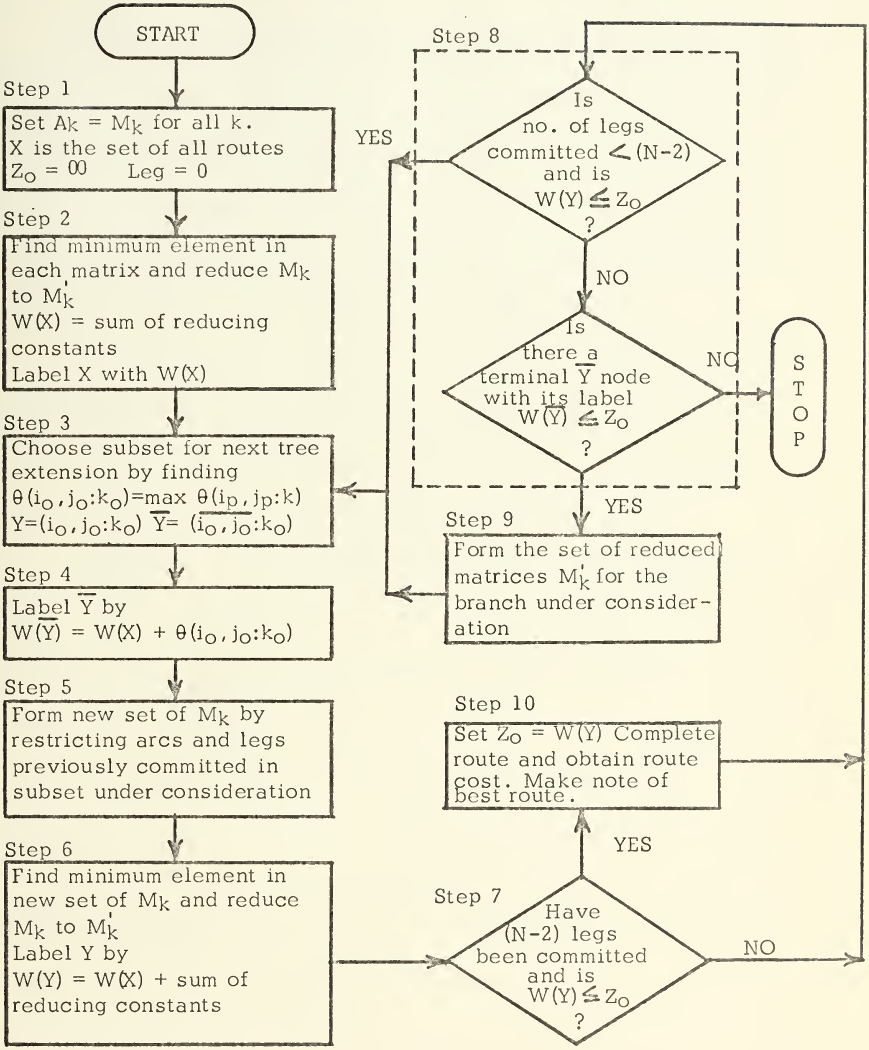


Figure 2

III. PROGRAMMING CONSIDERATIONS

The first decision that had to be made before programming of the algorithm began was what computer language would be most appropriate. Since one of the primary purposes of this project was to explore the feasibility of computerized solutions using the algorithm rather than to develop an efficient program for large-scale[3] problems, FORTRAN IV was chosen as the language due to its ease of application.

One of the important factors to consider for computer applications is requirement for storage space. The strategy used for selection of the branch point in Step 8 can have a direct effect on storage requirements. There are two basic strategies which may be used:

Strategy 1: Branch from the lowest bound. This strategy is the one used in the original algorithm [Ref. 1] and has the advantage that the total computation required to reach optimality is minimized in the sense that any branching performed is also that which must be performed under any alternate policy. Its primary disadvantage is that no terminal nodes are discarded and hence storage requirements may become excessive. In addition, it brings Step 9 of the algorithm into play more often which requires time to backtrack through the tree and set up the matrices for a further branch from the chosen node.

---

[3] Large-scale here is considered to be when the number of nodes, N, is greater than 20.

Strategy 2: Branch always from the latest Y node if a complete route has not been determined and discard nodes from storage that are no longer in contention for branch points or for the optimal route. This is known as a "branch to the right" policy. It has as its primary advantage that the amount of computer storage required is minimized since nodes are discarded when they are no longer required. Also, Step 9 of the algorithm will not be called upon as frequently as under Strategy 1.

Strategy 1 was originally employed, but for the few test cases considered, the number of iterations and time required to obtain the optimal route was in general greater than that required under Strategy 2 and hence the program presented uses Strategy 2.

As mentioned in Reference 1, a very useful feature of this routing algorithm is that one can stop at any point after the first complete route has been determined and have a feasible tour, although it may not be optimal. In the computer application of the algorithm, it may be the case that sufficient storage space or time required to reach the optimal solution may not be available. Hence, if one is willing to accept a suboptimal solution such as a solution below a given cost, this given cost could be input to the program and as soon as a solution that has a cost less than this amount has been found, computation can be halted. This may be found to be extremely useful when dealing with large-scale problems where to pay for sufficient computer time to reach the optimal solution might be prohibitive [Refs. 1 and 5]. Note that in test problem

Number 11, a solution within 4% of the known optimal solution was obtained in a very short period of time, but that nearly 300 minutes and 25,000 iterations later, the same solution was found and the optimal route had still not been located.

Although Reference 1 gives a modification to the basic algorithm for symmetric matrices, the modification was not incorporated in the program presented here.

For test problems 1 - 10 presented in Section V, storage requirements did not become excessive as will be discussed in more detail under Section VI on computational results.

COMPUTER PROGRAM FLOW

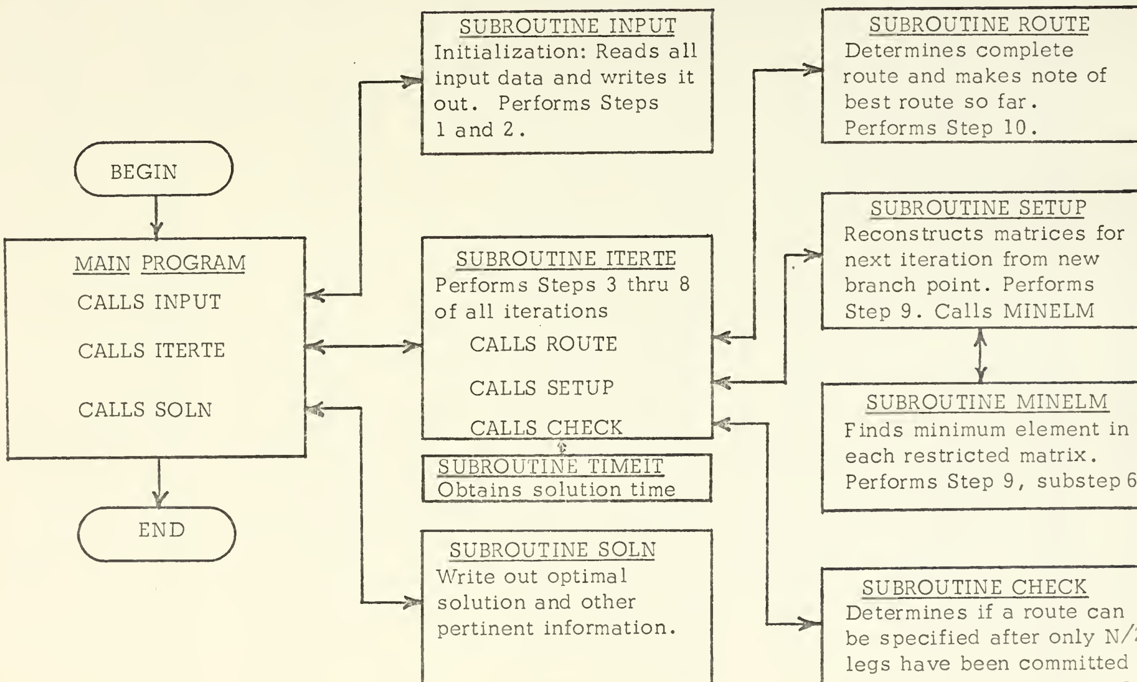


Figure 3

IV. THE COMPUTER PROGRAM

The computer program is entirely integer in nature except for the variables used in conjunction with the timing routine. A detailed description of the major variables used in the program may be found in Appendix D. Originally, the program was compiled using the FORTRAN G-level compiler and consisted of a main program where the major portion of all iterations was accomplished, and two subroutines, one used for Step 5 of the algorithm and the other for output.

It was noted that the FORTRAN H-level compiler generated an object code which was superior to the G-level compiler, particularly for extensive looping and arithmetic operations which were present in this type of program. An attempt was made to compile the identical program using this H-level compiler but the program size combined with its complexity was too large for the compiler to accommodate. At this point, the program was broken up into a main program and eight subroutines, all of which the H-level compiler could handle. The computer program flow along with a brief description of the subroutines is illustrated in Figure 3.

Maximum storage utilization was attained by specifying that nearly all variables be INTEGER*2. This meant that the principal iteration information for the tree which was maintained for purposes of being able to branch from any node was limited to numbers less than or equal to 32,767. This limitation applies to bounds on nodes and

number of iterations; hence node numbers, since node numbers are directly related to iterations. All right-hand nodes are labeled by positive numbers which identify them with the iteration on which they were obtained and likewise, left-hand nodes are labeled by negative numbers.

Another limitation of the program as presented is that the number of nodes be equal to or less than 20. The number of tours which can be expected to be obtained is limited to 30. The first tour is obtained by branching to the right immediately until a complete tour is specified which takes place on the (N-2)-nd iteration. All future tours must have cost equal to or less than the previous tour or they are not considered or counted as a tour for the purposes of the program.

All of the limitations discussed are limits of the computer program as presented and may be easily modified by changing the appropriate DIMENSION statements. Iteration information contained in the matrices YTAB and YBTAB which is used for constructing the branch point becomes the primary storage-limiting factor when the number of iterations is expected to be in the thousands. For 150 iterations, which was used for the first eight test problems, the entire program required 114,000 (114K) bytes of storage. Each increment of 100 iterations above the 150 used requires 1.8K bytes of storage and therefore 2500 iterations as used for test problem Number 9 required 42K more bytes which led to a program size of 156K.

For the typical traveling salesman problems discussed in the next section, the optimal route as expressed by the computer output has been adjusted to reflect the actual route which excludes the dummy node (N+1). Typical computer solution output for both a sequence-dependent case and a typical traveling salesman case may be found in Appendix B.

A timing routine used in Reference 4 is included in the program for purposes of obtaining actual problem solution times which excludes all input and output buffering times.

The program follows the algorithm step by step. Documentation is interspersed throughout to enable a casual reader to understand the basic program flow. The entire program may be found in Appendix E. Appendix C contains the make up of the computer card deck.

In order to provide dynamic allocation of storage space based upon the number of nodes in a given problem and the number of iterations desired, modifications to the basic program presented in Appendix E have been provided in Appendix F. Details on the specific changes are given in Appendix F. The primary advantage of these modifications is that the user does not have to change all of the variable specifications and dimension information cards in the 8 primary routines each time different values for N and ITS are used (N is the number of nodes; ITS is the number of iterations desired). Only the appropriate job control language (JCL) card which specifies the storage and time requirements for the execution of the program must be changed.

V. TEST PROBLEMS

A. SEQUENCE DEPENDENT PROBLEMS

The first three test cases were problems whose description places them into the class of sequence-dependent routing problems. Problems 1 through 3 were taken directly from DeHaemer [1]. Problems 1 and 2 consisted of matrices which were asymmetric. In problem 3, all matrices were symmetric. As was noted in Section III, the computer program does not provide for special treatment of symmetric matrices, but it was desirable to include symmetric matrices as test problems.

Problem No. 1

Suppose an itinerant salesman must be routed so that his travel expenses are minimized while visiting 5 different cities. He must complete a leg of his route on each of 4 consecutive days. Travel expenses vary as a function of the day on which the travel occurs. At certain times, no public transportation is available and the costs reflect the price of the available charter transportation. All possible costs have been tabulated for each of the 4 traveling days and are presented in Figure 4.

	M_1					M_2					M_3					M_4				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1	x	3	11	14	6	x	6	11	12	7	x	6	14	9	29	x	17	11	22	9
2	10	x	7	9	15	13	x	5	10	13	16	x	24	8	15	28	x	16	19	10
3	23	12	x	29	4	26	24	x	15	14	7	25	x	3	17	24	20	x	21	6
4	22	24	13	x	5	21	8	20	x	18	5	18	15	x	13	15	14	12	x	1
5	16	19	20	26	x	9	16	23	29	x	26	12	23	2	x	14	16	7	13	x

Problem No. 1: Initial Set of Cost Matrices

Figure 4

Problem No. 2

This problem has the same framework as problem 1 except that there are 6 different cities and thus there are 5 legs of the route. The matrices of all possible costs are tabulated in Figure 5.

M_1

	1	2	3	4	5	6
1	x	40	24	32	28	12
2	36	x	20	36	4	32
3	24	32	x	8	16	16
4	12	20	20	x	24	16
5	8	32	12	8	x	8
6	16	24	16	20	12	x

M_2

	1	2	3	4	5	6
1	x	10	6	8	7	3
2	9	x	5	9	1	8
3	6	8	x	2	4	4
4	3	5	5	x	6	4
5	2	8	3	2	x	2
6	4	6	4	5	3	x

M_3

	1	2	3	4	5	6
1	x	30	18	24	21	9
2	27	x	15	27	3	24
3	18	24	x	6	12	12
4	9	15	15	x	18	12
5	6	24	9	6	x	6
6	12	18	12	15	9	x

M_4

	1	2	3	4	5	6
1	x	20	12	16	14	6
2	18	x	10	18	2	16
3	12	16	x	4	8	8
4	6	10	10	x	12	8
5	4	16	6	4	x	4
6	8	12	8	10	6	x

M_5

	1	2	3	4	5	6
1	x	50	30	40	35	15
2	45	x	25	45	5	40
3	30	40	x	10	20	20
4	15	25	25	x	30	20
5	10	40	15	10	x	10
6	20	30	20	25	12	x

Problem No. 2: Initial Set of Cost Matrices

Figure 5

Problem No. 3

Figure 6 contains a set of four symmetric cost matrices from which a minimal cost route is desired.

	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1	x	3	11	14	6	x	6	11	12	7	x	3	14	9	29	x	17	11	9	22
2	3	x	7	9	15	6	x	5	10	13	3	x	24	8	15	17	x	16	10	19
3	11	7	x	29	4	11	5	x	15	14	14	24	x	7	17	11	16	x	4	21
4	14	9	29	x	5	12	10	15	x	18	9	8	7	x	5	9	10	4	x	1
5	6	15	4	5	x	7	13	14	18	x	29	15	17	5	x	22	19	21	1	x

Problem No. 3: Initial Set of Cost Matrices

Figure 6

These first three examples are discussed in more detail along with sample calculations in Ref. 1.

It would have been desirable to have larger test problems for which an optimal route was known. In order to avoid the lengthy hand computations involved in the setup and solution of a larger problem, it was thought that the typical closed-loop traveling salesman problems which have known optimal solutions and are abundant in the literature could provide additional test cases.

B. TRAVELING SALESMAN PROBLEMS

By appropriate modification of the input data, the typical closed-loop traveling salesman problem (hereafter referred to as TSP) can be solved by the program. It was necessary that the problem be structured in a manner such that the route would be closed as opposed to the open-ended route determined by the algorithm, visiting each node exactly once. Since the optimal route in a TSP is independent of the starting node, it was observed that the addition of one dummy node and hence one dummy leg attained the desired results. This can best be illustrated by an example.

Suppose the following matrix of costs between 4 nodes was given:

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & x & 2 & 6 & 3 \\ 2 & 5 & x & 1 & 4 \\ 3 & 2 & 3 & x & 5 \\ 4 & 1 & 7 & 2 & x \end{array}$$

N = 4

No. of legs = N - 1 = 3

It is assumed, as is usually the case in the TSP, that the matrix is the same for each leg. Consider the following set of four matrices which have one additional dummy node (node 5) besides the original 4 from above.

$$\begin{array}{c|ccccc|ccccc|ccccc|ccccc} & \multicolumn{5}{c|}{M_1} & \multicolumn{5}{c|}{M_2} & \multicolumn{5}{c|}{M_3} & \multicolumn{5}{c}{M_4} \\ & 1 & 2 & 3 & 4 & 5 & 1 & 2 & 3 & 4 & 5 & 1 & 2 & 3 & 4 & 5 & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & x & 2 & 6 & 3 & x & x & x & x & x & x & x & x & x & x & x & x & x & x & x & x \\ 2 & x & x & x & x & x & x & x & 1 & 4 & x & x & x & 1 & 4 & x & x & x & x & x & 5 \\ 3 & x & x & x & x & x & x & 3 & x & 5 & x & x & 3 & x & 5 & x & x & x & x & x & 2 \\ 4 & x & x & x & x & x & x & 7 & 2 & x & x & x & 7 & 2 & x & x & x & x & x & x & 1 \\ 5 & x & x & x & x & x & x & x & x & x & x & x & x & x & x & x & x & x & x & x & x \end{array}$$

N = 5

No. of legs = N-1 = 4

The number of nodes is now 5 and hence 4 legs are required to complete a route. Matrix M_1 is used to force the algorithm to choose leg one with an arc leading from node 1, to one of the other original nodes, nodes 2, 3, or 4, since all other arc choices on the first leg have prohibitive costs associated with them. Matrix M_4 is a dummy leg which is used to form a closed-loop. The only entries of significance in M_4 are those in the last column, column 5. These m_{i5}^4 values represent the costs of going from any node to node 1, since leg one began with an arc leading from node 1. Since the only "acceptable" values are $m_{25} = 5$, $m_{35} = 2$, and $m_{45} = 1$ as $m_{i5} = \infty$ for i = 1 and 5,

the optimal route will be forced to close on node 1 as desired. Matrices M_2 and M_3 are identical and are designed to prevent any arc from originating at node 1 or node 5 and to prevent any arc from terminating at node 1 or node 5, and therefore rows 1 and 5 and columns 1 and 5 have infinite values. Note that the dotted lines in M_2 and M_3 contain the original matrix less row 1 and column 1, as illustrated by the dotted lines in the original matrix.

The general pattern which emerges is that the matrix for leg 1 would contain all infinite values except for those arcs leading from node 1 to all the other original nodes. The last matrix would contain all infinite values except for the last column which would be the same as the first column of the original matrix with the infinite value below it. The intermediate matrices would be the same as the original matrix less row 1 and column 1 with an entire border of infinite values added to them.

With the above modifications, the following traveling salesman problems were solved as though they were sequence-dependent routing problems. (Only the original matrix is given.)

Problem No. 4 [Ref. 6]

	1	2	3	4	5
1	x	5	6	10	8
2	5	x	5	12	12
3	6	5	x	8	10
4	10	12	8	x	6
5	8	12	10	6	x

Problem No. 4: Initial Cost Matrix

Figure 7

Problem No. 5 [Ref. 6]

	1	2	3	4	5	6
1	x	4	3	7	7	6
2	4	x	2	5	7	7
3	3	2	x	5	6	6
4	7	5	5	x	3	5
5	7	7	6	3	x	3
6	6	7	6	5	3	x

Problem No. 5: Initial Cost Matrix

Figure 8

Problem No. 6 [Ref. 5]

	1	2	3	4	5	6
1	x	27	43	16	30	26
2	7	x	16	1	30	25
3	20	13	x	35	5	0
4	21	16	25	x	18	18
5	12	46	27	48	x	5
6	23	5	5	9	5	x

Problem No. 6: Initial Cost Matrix

Figure 9

Problem No. 7 [Ref. 2]

	1	2	3	4	5	6	7	8	9	10
1	x	51	55	90	41	63	77	69	0	23
2	50	x	0	64	8	53	0	46	73	72
3	30	77	x	21	25	51	47	16	0	60
4	65	0	6	x	2	9	17	5	26	42
5	0	94	0	5	x	0	41	31	59	48
6	79	65	0	0	15	x	17	47	32	43
7	76	96	48	27	34	0	x	0	25	0
8	0	17	0	27	46	15	84	x	0	24
9	56	7	45	39	0	93	67	79	x	38
10	30	0	42	56	49	77	72	49	23	x

Problem No. 7: Initial Cost Matrix

Figure 10

Problem No. 8 [Ref. 2]

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	x	57	72	15	66	49	0	53	28	60	60	65	12
2	0	x	0	82	40	24	31	4	21	59	33	59	27
3	92	35	x	98	80	57	67	0	48	84	86	77	26
4	77	76	64	x	67	0	36	94	70	63	29	0	46
5	74	95	14	63	x	14	47	24	98	0	0	24	80
6	96	5	4	0	44	x	86	54	28	36	22	41	73
7	99	76	44	92	35	36	x	25	35	0	33	37	42
8	93	73	37	73	76	73	94	x	0	92	59	52	58
9	24	70	91	94	60	8	73	52	x	0	94	81	65
10	67	0	53	23	0	51	77	66	11	x	52	86	21
11	19	95	0	50	79	84	79	37	45	8	x	57	0
12	74	0	29	92	13	54	78	61	46	69	40	x	29
13	60	43	25	42	15	19	0	87	75	53	52	67	x

Problem No. 8: Initial Cost Matrix

Figure 11

Problem No. 9 [Ref. 3]

	1	2	3	4	5	6	7	8	9	10
1	x	24	18	22	31	19	33	25	30	26
2	15	x	19	27	26	32	25	31	28	18
3	22	23	x	23	16	29	27	18	16	27
4	24	31	18	x	19	13	28	9	19	27
5	23	18	34	20	x	31	24	15	25	8
6	24	12	17	15	10	x	11	16	21	31
7	28	15	27	35	19	18	x	21	21	19
8	13	24	18	13	13	22	25	x	29	24
9	17	21	18	24	27	24	34	31	x	18
10	18	19	29	16	23	17	18	31	23	x

Problem No. 9: Initial Cost Matrix

Figure 12

Problem No. 10 [Ref. 7]

	1	2	3	4	5	6	7	8	9	10
1	x	28	57	72	81	85	80	113	89	80
2	28	x	28	45	54	57	63	85	63	63
3	57	28	x	20	30	28	57	57	40	57
4	72	45	20	x	10	20	72	45	20	45
5	81	54	30	10	x	22	81	41	10	41
6	85	57	28	20	22	x	63	28	28	63
7	80	63	57	72	81	63	x	80	89	113
8	113	85	57	45	41	28	80	x	40	80
9	89	63	40	20	10	28	89	40	x	40
10	80	63	57	45	41	63	113	80	40	x

Problem No. 10: Initial Cost Matrix

Figure 13

Problem No. 11 [Ref. 2]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	x	29	41	9	18	6	42	48	74	43	51	7	36	93	58	11	51	61	30	44
2	29	x	72	72	50	39	60	34	25	46	25	35	14	20	35	83	27	86	95	30
3	41	72	x	70	54	35	59	88	19	72	87	38	24	68	63	80	58	40	89	24
4	9	72	70	x	60	20	24	73	79	51	43	58	4	47	29	22	48	27	88	91
5	18	50	54	60	x	17	74	93	0	76	30	55	84	42	47	91	21	59	24	80
6	6	39	35	20	17	x	26	60	32	63	84	21	26	96	75	14	13	51	16	83
7	42	69	59	24	74	26	x	97	65	64	13	23	3	78	15	30	56	22	13	58
8	48	34	88	73	93	60	97	x	63	27	42	62	32	20	26	5	80	52	47	36
9	74	25	19	79	0	32	65	63	x	71	91	5	85	51	72	53	8	49	90	39
10	43	46	72	51	76	63	64	27	71	x	66	30	57	8	71	19	25	10	83	40
11	51	25	87	43	30	84	13	42	91	66	x	9	26	6	99	33	8	99	92	31
12	7	35	38	58	55	21	23	62	5	30	9	x	86	27	34	72	45	59	32	77
13	36	16	24	4	84	26	3	32	85	57	26	86	x	12	28	24	60	19	12	20
14	93	20	68	47	42	96	78	20	51	8	6	27	12	x	19	77	14	22	54	77
15	58	35	63	29	47	75	15	26	72	71	99	34	28	19	x	22	75	28	72	64
16	11	83	80	22	91	14	30	5	53	19	33	72	24	77	22	x	62	79	97	47
17	51	27	58	48	21	13	56	80	8	25	8	45	60	14	75	62	x	91	59	75
18	61	86	40	27	59	51	22	52	49	10	99	59	19	22	28	79	91	x	87	4
19	30	95	89	88	24	16	13	47	90	83	92	32	12	54	72	97	59	87	x	32
20	44	30	24	91	80	83	58	36	39	40	31	77	20	77	64	47	75	4	32	x

Optimal Solution: 1-12-11-17-6-16-8-15-7-19-5-9-3-20-18-10-
14-2-13-4-1

Optimal Route "Cost" = 246

Problem No. 11: Initial Cost Matrix

Figure 14

VI. COMPUTATIONAL RESULTS

Table I presents summary statistics for the eleven test problems considered. Optimal routes obtained verified the known results which are in the respective references from which the problems were taken, with the exception of test problem Number 7. Reference 2 indicates that the optimal route for this problem is as specified in the notes for Table I with an optimal route cost of 33. This program obtained the optimal route indicated in the table with a route cost of 28 which is 5 cost units superior to the previous known result.

The type of problem is either sequence-dependent (SD) or traveling salesman problem (TSP) as discussed in Section V. The number of complete tours obtained by the program is significant in that after the first tour is obtained by branching only to the right, a succeeding tour found must have a cost equal to or less than the best tour located so far in the computational procedure. It is somewhat representative of the "speed" of convergence towards the optimal solution. The number of iterations required is actually the number necessary to verify that the best route found by the program is the optimal route. The iteration number on which the optimal route is located is in general, far lower than the total number of iterations required for verification (note test problems Numbers 9 and 10).

Test problems Numbers 4, 5, and 10 have alternate routes indicated, but these routes are mirror images of one another and hence

Table I

SUMMARY STATISTICS

Program Compilation Time: Approx. 40 seconds

Test Problem Number	Type Prob[1]	Number of Nodes[2]	No. of Tours Obtained[3]	Number of Iterations[4]	Iter. when opt. found	Optimal "Cost"	Optimal Routes FM-TO-TO......	Running Time (Seconds)
1	SD	5	1	3	3	12	1-2-3-4-5	0.0333
2	SD	6	2	22	8	33	2-5-1-6-3-4	0.7255
3	SD	5	1	13	3	16	1-2-3-4-5	0.3195
4	TSP	6	2	13	4	32	1-2-3-4-5-1	0.3594
					10	32	1-5-4-3-2-1 (alternate)	
5	TSP	7	2	27	5	32	1-3-2-4-5-6-1	1.4310
					13	22	1-6-5-4-2-3-1 (alternate)	
6	TSP	7	3	46	31	63	1-4-3-5-6-2-1	2.0500
7	TSP	11	5	122	111	28+	1-10-2-7-6-4-8-3-9-5-1	14.567
8	TSP	14	2	56	31	20	1-13-7-10-5-11-3-8-9-6-4-12-2-1	14.523
9	TSP	11	7	2444	121	146	1-3-9-4-8-5-10-6-7-2-1	503.60
10	TSP	11	9	14931	356	378	1-7-6-8-9-10-5-4-3-2-1	3694.9
					519	378	1-2-3-4-5-10-9-8-6-7-1 (alternate)	
11	TSP	21	13	25000	not found 2624	best is 256	best route so far is 1-6-17-2-13-4-16-8-15-7-19-5-9-3-20-18-10-14-11-12-1	18,026

Notes:

1. SD is sequence dependent type problem: TSP is typical closed-loop traveling salesman problem.
2. If problem is of type TSP, number of actual nodes is one less than that recorded here.
3. Each tour had a cost equal to or less than the preceding tour.
4. Number of iterations required to verify optimal solution.

\+ Indicates optimal route differs from that reported in reference 2 which states that the route 1-9-5-6-4-7-10-2-3-8-1 with cost of 33 is optimal.

are identical. This fact is due to the symmetric nature of the input matrices combined with the fact that the matrices are the same for all legs of the route excluding the dummy legs. It would be desirable to eliminate consideration of any future route which would be an image of a route previously located, but this feature was not incorporated into the program. For the sequence-dependent symmetric case, test problem Number 3, only a single route is found as the matrices for each leg are symmetric, but different for each leg.

Since only a few cases were presented, it would be difficult to attempt to draw any conclusions with respect to expected time required for solution of a problem of given size. However, it was observed that the time required for a solution rises rapidly as the number of nodes increases as discussed in Reference 2. The computer storage requirement for test problems 1 through 9 was a moderate 154K, but problem Number 10 required 392K. Test problem Number 11 was run for approximately 300 minutes and 25,000 iterations which required 546K bytes of storage and the optimal solution was never reached. This matrix is symmetric and hence the number of iterations could be reduced by taking this fact into account.

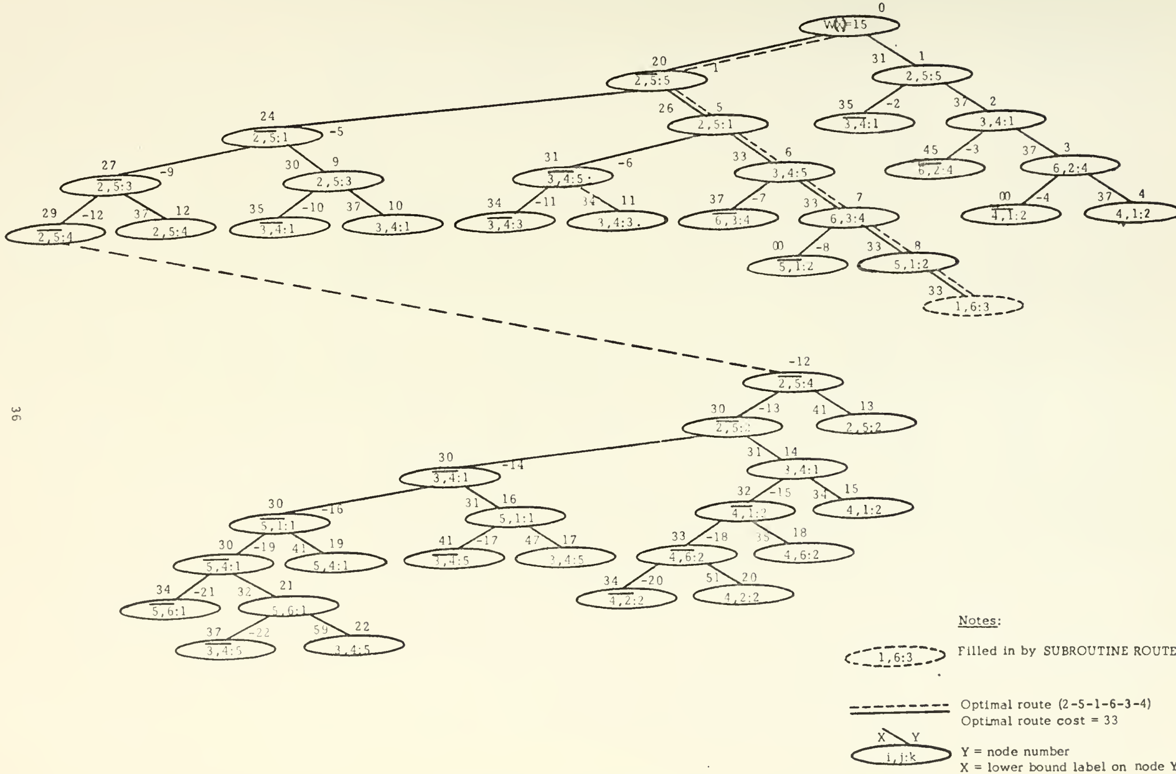
Test problems 7 and 8 terminated in just a few iterations but the zeroes in the matrix were placed somewhat strategically. In problems 9 and 10, the entries in the matrix are nearly all two digit numbers which are close to each other in magnitude and hence there is no clear-cut minimum route as in problems 7 and 8, and thus the number of iterations runs up into the thousands.

VII. CONCLUSIONS

The branch-and-bound algorithm and the computer program presented can successfully find the optimal route for a variety of sequence-dependent routing problems when the matrices of all possible costs for each leg of the route are known.

Although it is admitted that the computer program, as written in FORTRAN IV, may not be the most efficient for large-scale problems due to storage requirements and processing time, it does provide a basis for further programming effort using this algorithm. Although no attempt was made to delete nodes from storage once the node bound was observed to be above the current least upper bound on a complete route, larger scale problems would demand such reduction.

In the case of symmetric matrices, a programming method must be devised to delete consideration of arc (j,i) when arc (i,j) has been committed to a leg as this just leads to excessive computation and excessive iterations. It is recommended that a lower-level language, such as Assembly Language, be utilized to improve efficiency with respect to both time and storage requirements since the algorithm deals primarily with integer arithmetic operations.



COMPLETE COMPUTER SOLUTION TREE FOR TEST PROBLEM NUMBER 2

APPENDIX B

CASE NO. 2

EXAMPLE 2 FRCM REFERENCE 1

NUMBER OF NODES = 6

NUMBER OF LEGS = 5

TYPE PROBLEM: SEQUENCE-DEPENDENT

MATRIX M1

***	40	24	32	28	12
36	***	20	36	4	32
24	32	***	8	16	16
12	20	20	***	24	16
8	32	12	8	***	8
16	24	16	20	12	***

MATRIX M2

***	10	6	8	7	3
9	***	5	9	1	8
6	8	***	2	4	4
3	5	5	***	6	4
2	8	3	2	***	2
4	6	4	5	3	***

MATRIX M3

***	30	18	24	21	9
27	***	15	27	3	24
18	24	***	6	12	12
9	15	15	***	18	12
6	24	9	6	***	6
12	18	12	15	9	***

MATRIX M4

***	20	12	16	14	6
18	***	10	18	2	16
12	16	***	4	8	8
6	10	10	***	12	8
4	16	6	4	***	4
8	12	8	10	6	***

MATRIX M5

***	50	30	40	35	15
45	***	25	45	5	40
30	40	***	10	20	20
15	25	25	***	30	20
10	40	15	10	***	10
20	30	20	25	15	***

(SEE SOLUTION ON NEXT PAGE)

****

FEASIBLE TOUR NO. 2 IS DECLARED OPTIMAL

LEG	FROM	TO	COST
1	2	5	4
2	5	1	2
3	1	6	9
4	6	3	8
5	3	4	10

OPTIMAL ROUTE COST = 33

****

NUMBER OF ALTERNATE OPTIMAL TOURS = 0

NUMBER OF ITERATIONS REQUIRED = 22

TIME TO COMPUTE SOLUTION= 0.738816 SECONDS

ITERATION INFORMATION

		YTABLE					YBARTABLE	
NODE	FROM	WY	IO	JO	KO	TERM	WYBAR	TERM
1	0	31	2	5	5	0	20	0
2	1	37	3	4	1	0	35	0
3	2	37	6	2	4	0	45	0
4	3	37	4	1	2	0	32037	0
5	-1	26	2	5	1	0	24	0
6	5	33	3	4	5	0	31	0
7	6	33	6	3	4	0	37	0
8	7	33	5	1	2	0	32033	0
9	-5	30	2	5	3	0	27	0
10	9	37	3	4	5	0	35	0
11	-6	34	3	4	3	0	34	0
12	-9	37	2	5	4	0	29	0
13	-12	41	2	5	2	0	30	0
14	-13	31	3	4	1	0	30	0
15	14	34	4	1	2	0	32	0
16	-14	31	5	1	1	0	30	0
17	16	47	3	4	5	0	41	0
18	-15	35	4	6	2	0	33	0
19	-16	41	5	4	1	0	30	0
20	-18	51	4	2	2	0	34	0
21	-19	32	5	6	1	0	34	0
22	21	59	3	4	5	0	37	0

CASE NO. 6

EXAMPLE FRCM ARTICLE BY LITTLE AND OTHERS IN OR JOURNAL 1963

NUMBER OF NODES = 7

NUMBER OF LEGS = 6

TYPE PROBLEM: TRAVELING SALESMAN CLOSED-LOOP

MATRICES ARE SAME FOR LEGS 2 THRU (N-2) AND ARE AS FOLLOWS:

I/J=	1	2	3	4	5	6	7
1	9999	9999	9999	9999	9999	9999	9999
2	9999	9999	16	1	30	25	9999
3	9999	13	9999	35	5	0	9999
4	9999	16	25	9999	18	18	9999
5	9999	46	27	48	9999	5	9999
6	9999	5	5	9	5	9999	9999
7	9999	9999	9999	9999	9999	9999	9999

****

FEASIBLE TOUR NO. 3 IS DECLARED OPTIMAL

LEG	FROM	TO	COST
1	1	4	16
2	4	3	25
3	3	5	5
4	5	6	5
5	6	2	5
6	2	1	7

OPTIMAL ROUTE COST = 63

****

NUMBER OF ALTERNATE OPTIMAL TOURS = 0

NUMBER OF ITERATIONS REQUIRED = 46

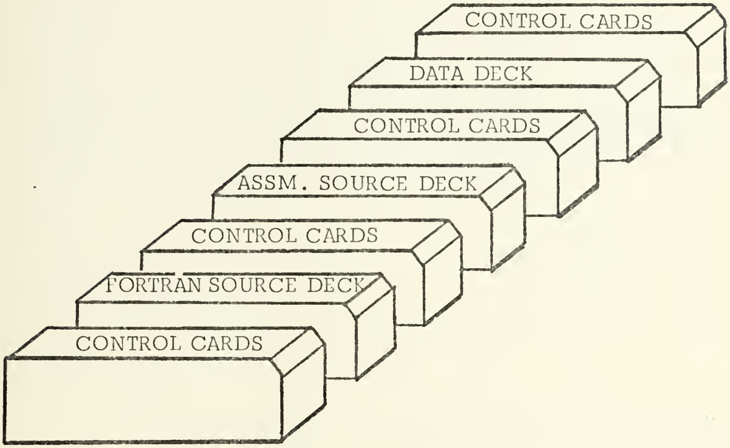
TIME TO COMPUTE SOLUTION= 2.023424 SECONDS

ITERATION INFORMATION

		YTABLE						YBARTABLE	
NODE	FROM	WY	IO	JO	KO	TERM		WYBAR	TERM
1	0	39	1	4	1	0		33	0
2	1	83	3	6	3	0		44	0
3	2	83	4	3	2	0		32083	0
4	3	88	2	5	5	0		99	0
5	4	88	6	2	4	0		32088	0
6	-1	38	2	7	6	0		38	0
7	6	52	6	2	5	0		46	0
8	7	94	1	5	1	0		65	0
9	-2	60	3	6	4	0		49	0
10	9	65	2	3	3	0		71	0
11	10	65	4	2	2	0		32065	0
12	11	65	6	5	5	0		10059	0
13	-6	43	5	7	6	0		46	0
14	13	50	1	6	1	0		44	0
15	14	65	3	5	5	0		63	0
16	15	74	2	4	3	0		80	0
17	-7	89	3	6	2	0		51	0
18	-8	83	1	3	1	0		10021	0
19	-9	101	3	6	5	0		54	0
20	-13	54	3	7	6	0		49	0
21	20	59	6	3	5	0		65	0
22	21	76	2	4	2	0		74	0
23	-14	45	1	2	1	0		60	0
24	23	63	2	4	2	0		60	0
25	24	68	4	6	3	0		70	0
26	-15	32069	2	4	3	0		75	0
27	-17	83	3	6	3	0		56	0
28	-19	56	2	7	6	0		59	0
29	28	56	6	2	5	0		64	0
30	29	63	5	6	4	0		32051	0
31	30	63	4	3	2	0		32063	0
32	-20	58	1	6	1	0		50	0
33	32	66	4	7	6	0		10033	0
34	-23	62	1	3	1	0		10016	0
35	34	79	3	6	2	0		75	0
36	-24	69	2	3	2	0		69	0
37	-27	32080	3	6	4	0		61	0
38	-28	59	5	7	6	0		67	0
39	38	70	4	2	2	0		61	0
40	-32	51	1	2	1	0		53	0
41	40	69	2	4	2	0		66	0
42	-37	61	1	6	1	0		65	0
43	42	107	3	5	3	0		74	0
44	-39	69	4	6	2	0		68	0
45	-40	58	1	5	1	0		66	0
46	45	66	5	6	2	0		80	0

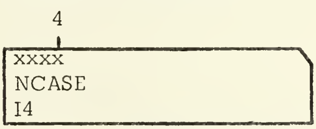
APPENDIX C

COMPOSITION OF COMPUTER CARD DECK



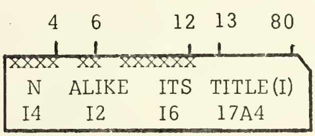
DATA DECK MAKEUP

Variable
Format



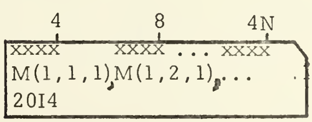
Card Type 1:
First card in Data Deck

Variable
Format



Card Type 2:
Second card in Data Deck and first card of each succeeding case

Variable
Format



Card Type 3:
Third card on until all matrices have been defined. N(N-1) cards for each case; input matrix for leg 1 first, then leg 2, etc., row by row.

Note: All values are integers and must be right-justified in format field specified.

APPENDIX D

DESCRIPTION OF VARIABLES USED IN PROGRAM

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

 PROGRAM CONSISTS OF ALL INTEGER VARIABLES EXCEPT TIMEX
 WHICH IS USED IN CONJUNCTION WITH THE TIMING ROUTINE

 PROGRAM LIMITATIONS :  20 NODES TOTAL INCLUDING THE
 AUGMENTED NODE;  NUMBER OF ITERATIONS IS LIMITED BY THE
 NUMBER SPECIFIED BY THE FIRST INDEX OF VARIABLE MATRICES
 YTAB AND YBTAB IN THE DIMENSION STATEMENTS AND MUST BE
 EQUAL TO OR LARGER THAN THE LARGEST VALUE OF THE VARIABLE
 ITS FOR ANY DATA SET IN THE DATA DECK

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


 INPUT DECK REQUIREMENTS :   CC = CARD COLUMN

 CARD 1: FORMAT(I4)    CC 1-4
     NCASE=NUMBER OF CASES TO BE PROCESSED ON THIS RUN

 CARD 2: FORMAT(I4,I2,I6,17A4)

     N=NUMBER OF NODES (FORMAT I4)    CC 1-4

     ALIKE=1 IF ENTRIES IN MATRIX ARE SAME FOR EACH LEG
          =0 IF ENTRIES IN MATRIX ARE DIFFERENT FOR EACH
           LEG
           (FORMAT I2)    CC 5,6

     ITS = MAXIMUM NUMBER OF ITERATIONS DESIRED(FORMAT I6)
           (FORMAT I6)     CC 7-12

     TITLE(I) = A  HEADING FOR EACH INDIVIDUAL PROBLEM
                (FORMAT 17A4)    CC 13-80


 CARD 3 THRU END OF DATA SET: FORMAT(20I4)
    M(I,J,K) = WORKING SET OF MATRICES:  MATRICES ARE
               LOADED ONE ROW AT A TIME
               (MATRIX FOR FIRST LEG, SECOND LEG, ETC.)

 ALL ABOVE DATA IN I FORMAT MUST BE RIGHT JUSTIFIED
 IN FIELD SPECIFIED

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

 NCASE = NUMBER OF CASES TO BE PROCESSED ON ONE COMPUTER
         RUN

 N = NUMBER OF NODES

 ITS = MAXIMUM NUMBER OF ITERATIONS DESIRED

 ALIKE = 0 IF PROBLEM IS SEQUENCE DEPENDENT TYPE PROBLEM
 ALIKE = 1 IF TRAVELING SALESMAN TYPE PROBLEM

 L = N-1 = NUMBER OF LEGS FOR ROUTE BETWEEN N NODES

 LEGREQ = N-2 = NUMBER OF LEGS REQUIRED TO DETERMINE ROUTE

 COST(K) = COST OF GOING FROM NODE FM(K) TO NODE TO(K)
           ON K-TH LEG OF ROUTE

 TCOST(TOUR) = TOTAL COST OF TOUR NUMBER (TOUR)
```

BEST(K,J) = MATRIX CONTAINING BEST TOUR AT ANY STAGE OF SOLUTION AFTER FIRST SUBOPTIMAL TOUR HAS BEEN FOUND(K=LEG OF ROUTE)
J=1 IS LEG OF ROUTE
=2 IS NODE FROM WHICH LEG K BEGINS
=3 IS NODE WHICH ENDS LEG K
=4 IS THE COST TO GO FROM J=2 TO J=3

BEST(N,1) = NUMBER OF LEAST COST TOUR DETERMINED AT ANY POINT AFTER FIRST TOUR IS LOCATED

BEST(N,2) = COST OF TOUR NUMBER BEST(N,1)

LEGCOM(K) = K IF LEG COMMITTED
= 0 IF LEG UNCOMMITTED

FM(K) = NODE OF DEPARTURE ON K-TH LEG OF ROUTE

TO(K) = NODE OF ARRIVAL ON K-TH LEG OF ROUTE

ARCCOM (I,J) = 100 IF NEITHER I NOR J ARE ON A COMMITTED LEG
=-99 IF EITHER NODE I OR NODE J IS ON A COMMITTED LEG

STEP = STEP NUMBER OF ALGORITHM

ITER = ITERATION NUMBER

TOUR = NUMBER OF TOUR FOUND BY ALGORITHM, EACH TOUR HAVING A COST WHICH IS LESS THAN OR EQUAL TO THE PRECEEDING TOUR

M(I,J,K) = WORKING SET OF MATRICES
= COST (OR OTHER VARIABLE TO BE MINIMIZED) OF GOING FROM NODE I TO NODE J ON K-TH LEG OF ROUTE

A(I,J,K) = ORIGINAL M(I,J,K) = PERMANENT FILE OF ALL INPUT MATRICES

MINEL(K) = MINIMUM ELEMENT IN MATRIX K WHEN LEG K IS UNCOMMITTED EXCLUDING ROWS AND/OR COLUMNS ASSOCIATED WITH NODES ON COMMITTED LEGS

MIN(K) = CURRENT MINIMUM ELEMENT IN MATRIX K WHEN LEG K IS UNCOMMITTED (USED DURING SEARCH FOR MINEL(K))

IK(K) = ROW CONTAINING MINEL(K)

JK(K) = COLUMN CONTAINIMG MINEL(K)

THETA = MAXIMUM OF THE SECOND SMALLEST ELEMENTS IN ALL RESTRICTED MATRICES FOR UNCOMMITTED LEGS

MAXEL = CURRENT THETA IN THE DO-LOOP

MAXLEG = LEG FROM WHICH THETA WAS OBTAINED

IO = IK(MAXLEG)

JO = JK(MAXLEG)

LEG = NCOM = CURRENT NUMBER OF LEGS COMMITTED

WX=THE LOWER BOUND LABEL ATTACHED TO THE TREE FOR NODE X

WY=THE LOWER BOUND LABEL ATTACHED TO THE Y NODE OF TREE

WYBAR = THE LOWER BOUND LABEL ATTACHED TO THE YBAR NODE OF THE TREE

```
ZO = A LARGE NUMBER ORIGINALLY AND REMAINS AN UPPER BOUND
     ON THE OBJECTIVE FUNCTION

X(K) = AN ARRAY USED FOR DETERMINING THE FINAL LEG OF
       THE ROUTE AND NODES ON THIS LEG


INDEX = NODE NUMBER FROM WHICH TO BRANCH
        IS POSITIVE IF BRANCH IS TO BE FROM A Y NODE:
        IS NEGATIVE IF BRANCH IS TO BE FROM A YBAR NODE

YTAB(I,J) = A MATRIX CONTAINING INFORMATION ABOUT Y NODES
            (I) IN COLUMN J WHERE I = ITERATION WHICH
            GENERATED THE NODE
     J = 1 IS THE NODE NUMBER
       = 2 IS THE NODE FROM WHICH BRANCH WAS MADE
       = 3 IS LOWER BOUND LABEL ON NODE Y
       = 4 IS THE NODE OF DEPARTURE
       = 5 IS THE NODE OF ARRIVAL
       = 6 IS THE LEG OF ROUTE
       = 7 IS ZERO WHEN THE NODE IS NOT TERMINAL
              ONE WHEN THE NODE IS A TERMINAL NODE

YBTAB(I,J) = A MATRIX CONTAINING INFORMATION ABOUT YBAR
             NODES (I) IN COLUMN J WHERE I = ITERATION
             WHICH GENERATED THE NODE AND IS FOUND IN
             THE MATRIX YTAB(I,1)
     J = 1 IS THE LOWER BOUND LABEL ON YBAR NODE
     J = 2 (SAME AS FOR YTAB(I,7) )
        (NOTE:  THE NODE NUMBER IS THE NEGATIVE OF THE
        CORRESPONDING Y NODE FOR THE SAME ITERATION, I.)


BB, G, FROM, NCOM : ALL ARE VARIABLES USED IN RECON-
    STRUCTING MATRICES WHEN NODE FROM WHICH BRANCH IS TO
    OCCUR IS NOT NODE OF PREVIOUS STEP 6

KEY : IS A VARIABLE USED TO CONTROL FLOW OF PROGRAMMING
      THROUGH ALGORITHM TO AVOID ADDITIONAL DUPLICATE
      CODE WHICH WOULD BE REQUIRED

GOLF, DEL : VARIABLES USED TO DENOTE CURRENT ROW OR
    COLUMN OF ARCCOM MATRIX WHICH MAY BE USED FOR
    NOT CONSIDERING CERTAIN ELEMENTS OF THE MATRICES

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

FUNCTION SUBPROGRAMS USED :

 MINO - FINDS MINIMUM OF 2 OR MORE INTEGER*4 ARGUMENTS
        AND ASSIGNS A FUNCTIONAL INTEGER VALUE

 MAXO - FINDS MAXIMUM OF 2 OR MORE INTEGER*4 ARGUMENTS
        AND ASSIGNS A FUNCTIONAL INTEGER VALUE
```

APPENDIX E

COMPUTER PROGRAM LISTING

```
*********************************************************
*                                                       *
*                                                       *
*     A COMPUTER PROGRAM FOR THE SOLUTION               *
*     OF SEQUENCE-DEPENDENT ROUTING PROBLEMS            *
*     USING A BRANCH-AND-BOUND ALGORITHM                *
*                                                       *
*                                                       *
*********************************************************

C ***************************************************************

      IMPLICIT INTEGER*2(A-Z)
      REAL*4 TIMEX
      INTEGER*4 M(20,20,19),THETA,MAXEL,MINEL(20),MIN(20),
     1TITLE(17),ZC,WY
      DIMENSION A(20,20,19),COST(20),LEGCOM(20),TCOST(30)
     1FM(20),TO(2C),X(20),ARCCOM(20,20),BEST(20,4)
     2YTAB(2500,7),YBTAB(2500,2),IK(20),JK(20)
      COMMON TIMEX,M,THETA,MAXEL,MINEL,MIN,TITLE,ZO,WY,A,
     1COST,LEGCOM,FM,TO,X,ARCCOM,BEST,YTAB,YBTAB,IK,JK,TOUR,
     2AA,N,L,ALIKE,LEGREQ,STEP,ITER,DEL,WX,MAXLEG,LEG,WYBAR,
     3IO,JO,KC,ITS,TCOST,INDEX

    1 FORMAT (I4)
    2 FORMAT (I4,I2,I6,17A4)

      READ(5,1) NCASE

      DO 2000 AA = 1,NCASE

C   READ INPUT PARAMETERS

      READ(5,2) N,ALIKE,ITS,(TITLE(I),I=1,17)


      CALL INPUT

      CALL ITERTE (&2000)

C   OPTIMALITY REACHED : PROCESS A NEW CASE OR TERMINATE

      CALL SOLN

 2000 CONTINUE
      STOP
      END
```

```
      SUBROUTINE INPUT

      IMPLICIT INTEGER*2(A-Z)
      REAL*4 TIMEX
      INTEGER*4 M(20,20,19),THETA,MAXEL,MINEL(20),MIN(20),
     1TITLE(17),ZC,WY
      DIMENSION A(20,20,19),COST(20),LEGCOM(20),TCOST(30)
     1FM(20),TO(20),X(20),ARCCOM(20,20),BEST(20,4)
     2YTAB(2500,7),YBTAB(2500,2),IK(20),JK(20)
      COMMON TIMEX,M,THETA,MAXEL,MINEL,MIN,TITLE,ZO,WY,A,
     1COST,LEGCOM,FM,TO,X,ARCCOM,BEST,YTAB,YBTAB,IK,JK,TOUR,
     2AA,N,L,ALIKE,LEGREQ,STEP,ITER,DEL,WX,MAXLEG,LEG,WYBAR,
     3IO,JO,KC,ITS,TCOST,INDEX
      DATA HEAD/1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
     118,19,20,21,22,23,24/

    2 FORMAT (20I4)
    4 FORMAT (1H1)
    5 FORMAT('0',10X,'CASE NO.',I3,//,11X,17A4,//11X,'NUMBE'
     1,'R OF NODES = ',I2,//,11X,'NUMBER OF LEGS = ',I2)
    6 FORMAT('0',10X,'TYPE PROBLEM:  SEQUENCE-DEPENDENT')
    7 FORMAT('0',10X,'TYPE PROBLEM:  TRAVELING SALESMAN',
     1' CLOSED-LOOP')
    8 FORMAT('0',15X,'MATRIX M1',14X,'MATRIX M2',14X,
     1'MATRIX M3',14X,'MATRIX M4',14X,'MATRIX M5')
    9 FORMAT(/,11X,6I3,5X,6I3,5X,6I3,5X,6I3,5X,6I3)
   10 FORMAT('0',20X,'MATRIX M 1',20X,'MATRIX M 2',20X,
     1'MATRIX M 3',20X,'MATRIX M 4')
   11 FORMAT(/,10X,5I5,5X,5I5,5X,5I5,5X,5I5)
   33 FORMAT(11X,I2,2X,22I5)
   37 FORMAT('0',10X,'MATRICES ARE SAME FOR LEGS 2 THRU ',
     1'(N-2) AND ARE AS FOLLOWS:'//,11X,'I/J=',22I5)
   39 FORMAT('0')

C  INITIALIZATION

      DO 101 I=1,N
      COST(I) = 0
      LEGCOM(I) = 0
      FM(I) = 0
      TO(I) = 0
      DO 101 J=1,N
  101 ARCCOM(I,J) = 100
      DEL = 1
      ITER = 0
      TOUR = 0
      LEG = 0
      L = N-1
      LEGREQ = N - 2

C  NOTE: VALUES OF 32000 IN THE PROGRAM ARE USED TO
C  INDICATE INFINITE VALUES

      ZO = 32000

      STEP = 1

C  READ INPUT MATRICES FOR ALL L LEGS

      IF (ALIKE.EQ.1) GO TO 104
      DO 103 K=1,L
      DO 103 I=1,N
  103 READ(5,2) (M(I,J,K),J=1,N)
      GO TO 110
  104 DO 105 K=1,2
      DO 105 I=1,N
  105 READ(5,2) (M(I,J,K),J=1,N)
```

```
      DO 106 K=3,LEGREQ
      DO 106 I=1,N  .
      DO 106 J=1,N
  106 M(I,J,K) = M(I,J,K-1)
      DO 107 I=1,N
  107 READ(5,2) (M(I,J,L),J=1,N)
C  WRITE OUT INPUT

  110 WRITE(6,4)
      IF(ALIKE.EQ.0) WRITE(6,6)
      IF(ALIKE.EQ.1) WRITE(6,7)
      WRITE(6,5) AA,(TITLE(I),I=1,17),N,L
      IF(N.NE.5.AND.N.NE.6.OR.ALIKE.EQ.1) GO TO 113
      IF(N.EQ.5) WRITE(6,10)
      IF(N.EQ.6) WRITE(6,8)
      DO 111 I=1,N
      IF(N.EQ.5) WRITE(6,11) ((M(I,J,K),J=1,N),K=1,L)
  111 IF(N.EQ.6) WRITE(6,9) ((M(I,J,K),J=1,N),K=1,L)
      GO TO 131
  113 WRITE(6,37)(HEAD(I),I=1,N)
      WRITE(6,39)
      DO 117 I=1,N
  117 WRITE(6,33) I,((M(I,J,K),J=1,N),K=2,2)
  131 CONTINUE

C START TIMER

      CALL TIMEIT (0,TIMEX)

C  CREATE COPY OF ORIGINAL DATA IN MATRIX A

  112 DO 132 K=1,L
      DO 132 I=1,N
      DO 132 J=1,N
  132 A(I,J,K) = M(I,J,K)

C  STEP TWO
C  FIND MINIMUM ELEMENT IN EACH MATRIX

      DO 200 K=1,L
      IK(K) = 0
      JK(K) = 0
      MIN(K) = 32000
      DO 190 I=1,N
      DO 190 J =1,N
      IF(I.EQ.J) GO TO 190
      MINEL(K) = MINO (MIN(K),M(I,J,K))
  116 IF (MINEL(K).GE.MIN(K)) GO TO 190
      IK(K) = I
      JK(K) = J
      MIN(K) = MINEL(K)
  190 CONTINUE
  200 CONTINUE

C  REDUCE MATRICES

      DO 212 K=1,L
      DO 212 I=1,N
      DO 212 J=1,N
      IF(I.EQ.J) GO TO 212
      M(I,J,K) = M(I,J,K) - MINEL(K)
  212 CONTINUE

C  LABEL NODES

      WX = 0
      DO 230 K=1,L
  230 WX = WX + MINEL(K)
      RETURN
      END
```

```
      SUBROUTINE ITERTE (*)

      IMPLICIT INTEGER*2(A-Z)
      REAL*4 TIMEX
      INTEGER*4 M(20,20,19),THETA,MAXEL,MINEL(20),MIN(20),
     1TITLE(17),ZC,WY
      DIMENSION A(20,20,19),COST(20),LEGCOM(20),TCOST(30)
     1FM(20),TO(20),X(20),ARCCOM(20,20),BEST(20,4)
     2YTAB(2500,7),YBTAB(2500,2),IK(20),JK(20)
      COMMON TIMEX,M,THETA,MAXEL,MINEL,MIN,TITLE,ZO,WY,A,
     1COST,LEGCOM,FM,TO,X,ARCCOM,BEST,YTAB,YBTAB,IK,JK,TOUR,
     2AA,N,L,ALIKE,LEGREQ,STEP,ITER,DEL,WX,MAXLEG,LEG,WYBAR,
     3IO,JO,KO,ITS,TCOST,INDEX

   12 FORMAT('0',10X,'MAXIMUM ARRAY STORAGE SPACE EXCEEDED:'
     1,' ITER=',I6,' AND MATRICES YTAB AND YBTAB ARE DIMEN',
     2'SIONED FOR',I7' ITERATIONS.',//,11X,'NUMBER OF ',
     3'TOURS OBTAINED =',I3,'. BEST VALUE SO FAR IS ',I5,
     4' FOR TOUR NUMBER',I3,'.')
   15 FORMAT('0',10X,'THE INFORMATION OBTAIN BY THE PROGRAM'
     1,' SO FAR IS PRINTED OUT BELOW, SOLELY FOR REFERENCE',
     2'.  BEST ROUTE HAS NOT BEEN FOUND.')

C  STEP THREE  - - - ITERATION PROCEDURE
C                    CONSISTS OF STEPS THREE THRU EIGHT

      LABEL = 1
  500 ITER = ITER + 1

C  MAXIMUM ARRAY STORAGE SPACE EXCEEDED: ERROR

      IF(ITER.GT.ITS) CALL TIMEIT (-1,TIMEX)
      IF(ITER.GT.ITS) WRITE(6,12)ITER,ITS,TOUR,BEST(N,2),
     1BEST(N,1)
      IF(ITER.GT.ITS) GO TO 840

      MAXEL = -10
      MAXLEG = 0
      DO 512 K=1,L
      IF(K.EQ.LEGCOM(K)) GO TO 512
      MIN(K) = 32000
      MINEL(K) = -1
      DO 509 I=1,N
      DO 508 J=1,N
      IF(I.EQ.J) GO TO 508
      IF (K.EQ.L) GO TO 505
      IF(K.NE.LEGCOM(K+1)-1) GO TO 505
      IF(K.EQ.LEGCOM(K-1)+1) GO TO 510
      IF(J.NE.FM(K+1)) GO TO 508
      IF(ARCCOM(I,DEL).LT.100) GO TO 508
      GO TO 504
  505 IF(K.EQ.1) GO TO 506
      IF(K.NE.LEGCOM(K-1)+1) GO TO 506
      IF(I.NE.TO(K-1)) GO TO 508
      IF(ARCCOM(DEL,J).LT.100) GO TO 508
      GO TO 504
  506 IF(ARCCOM(I,J).NE.100) GO TO 508
      IF(ARCCOM(J,I).NE.100) GO TO 508
  504 IF (I.EQ.IK(K).AND.J.EQ.JK(K)) GO TO 508
      MINEL(K) = MINO(MIN(K),M(I,J,K))
      IF(MINEL(K).GE.MIN(K)) GO TO 508
      MIN(K) = MINEL(K)
  508 CONTINUE
  509 CONTINUE
  510 IF(MINEL(K).EQ.-1) MINEL(K) = 32000

      THETA = MAXC(MAXEL,MINEL(K))
      IF(THETA.LE.MAXEL) GO TO 512
      MAXLEG = K
      MAXEL = THETA
  512 CONTINUE
```

```
C  STEP FOUR - - ITERATION PROCEDURE
C  LABEL YBAR BY WYBAR

      WYBAR = WX + THETA
      LEGCCM(MAXLEG) = MAXLEG
      FM(MAXLEG) = IK(MAXLEG)
      TO(MAXLEG) = JK(MAXLEG)
      JO = TO(MAXLEG)
      IO = FM(MAXLEG)
      COST(MAXLEG) = A(IO,JO,MAXLEG)

      YBTAB(ITER,1) = WYBAR
      YBTAB(ITER,2) = 1

      YTAB(ITER,1) = ITER
      IF(ITER.EQ.1) YTAB(ITER,2) = 0
      IF(ITER.GT.1) YTAB(ITER,2) = INDEX
      YTAB(ITER,4) = IO
      YTAB(ITER,5) = JO
      YTAB(ITER,6) = MAXLEG
      YTAB(ITER,7) = 1

C  IF LEG JUST DETERMINED ALLOWS A ROUTE TO BE SPECIFIED,
C  GO TO STEP 7

      IF(LEG.EQ.N-3) YTAB(ITER,3) = WX
      IF(LEG.EQ.N-3) GO TO 700

C  STEP FIVE - - ITERATION PROCEDURE
C  DELETION OF ARCS NOT POSSIBLE

      DO 513 I=1,N
      DO 513 J=1,N
      IF(I.EQ.J) GO TO 513
      IF((I.EQ.IO.OR.I.EQ.JO.OR.J.EQ.IO.OR.J.EQ.JO).AND.
     1ARCCCM(I,J).GT.99) ARCCOM(I,J) = -99
  513 CONTINUE

      GOLF = 1
  515 DEL = GOLF
      DO 516 F=1,L
      IF (FM(F).EQ.DEL) GOLF = GOLF + 1
      IF (FM(F).EQ.DEL) GO TO 515
      IF (TO(F).EQ.DEL) GOLF = GOLF + 1
  516 IF (TO(F).EQ.DEL) GO TO 515

C  STEP SIX - - ITERATION PROCEDURE
C  FIND MINIMUM ELEMENT IN EACH MATRIX K WHERE LEG K
C  IS UNCOMMITTED

      DO 630 K=1,L
      KEY = 0
      IK(K) = 0
      JK(K) = 0
      IF (K.EQ.LEGCOM(K)) GO TO 630
      MINEL(K) = -1
  602 MIN(K) = 32000
      DO 629 I=1,N
      DO 629 J=1,N
      IF(I.EQ.J) GO TO 621
      IF (K.EQ.1.AND.LEGCOM(2).EQ.0) GO TO 605
      IF (K.EQ.1) GO TO 606
      IF (K.EQ.L.AND.LEGCOM(L-1).EQ.0) GO TO 605
      IF (K.EQ.L) GO TO 610
      IF(K.EQ.LEGCOM(K+1)-1.AND.K.NE.LEGCOM(K-1)+1)GOTO 606
      IF(K.EQ.LEGCOM(K-1)+1.AND.K.NE.LEGCOM(K+1)-1)GOTO 610
      IF(K.EQ.LEGCCM(K+1)-1.AND.K.EQ.LEGCOM(K-1)+1)GOTO 609
```

```
  605 IF (ARCCOM(I,J).LT.100) GO TO 621
      IF (KEY.EQ.0) GO TO 620
      IF (KEY.EQ.1) GO TO 622
  606 IF (J.NE.FM(K+1)) GO TO 621
      IF (ARCCOM(I,DEL).LT.100) GO TO 621
      IF (KEY.EQ.0) GO TO 620
      IF (KEY.EQ.1) GO TO 622
  609 IF(I.EQ.TO(K-1).AND.J.EQ.FM(K+1).AND.KEY.EQ.0)GOTO620
      IF(I.EQ.TO(K-1).AND.J.EQ.FM(K+1).AND.KEY.EQ.1)GOTO622
      GO TO 621
  610 IF (I.NE.TO(K-1)) GO TO 621
      IF (ARCCOM(DEL,J).LT.100) GO TO 621
      IF (KEY.EQ.1) GO TO 622
  620 MINEL(K) = MINO(MIN(K),M(I,J,K))
      IF (MINEL(K).GE.MIN(K)) GO TO 621
      IK(K) = I
      JK(K) = J
      MIN(K) = MINEL(K)
  621 IF (KEY.EQ.1) GO TO 629
      IF (I.NE.N) GO TO 629
      IF (J.NE.N) GO TO 629

C  REDUCE MATRICES

      KEY = 1
      GO TO 602
  622 IF(I.EQ.J) GO TO 629
      M(I,J,K) = M(I,J,K) - MINEL(K)
  629 CONTINUE
  630 CONTINUE

C  LABEL Y BY WY

      WY = WX
      DO 635 K=1,L
      IF(K.EQ.LEGCOM(K)) GO TO 635
      WY = WY + MINEL(K)
  635 CONTINUE
      YTAB(ITER,3) =  WY

C  STEP = 7 - - INCREMENT NUMBER OF LEGS COMMITTED AND
C               DETERMINE WHAT STEP IS NEXT

  700 LEG = LEG + 1
      IF(LEG.NE.N/2) GO TO 720
      CALL CHECK(&720)
      CALL ROUTE (&800,&840)
  720 IF(LEG.GE.LEGREQ.AND.WY.LE.ZO) CALL ROUTE(&800,&840)
      IF(LEG.GE.LEGREQ.AND.WY.GT.ZO) GO TO 799
      IF(LEG.LT.LEGREQ.AND.WY.LE.ZO) GO TO 850


C  STEP 8 - SELECT NODE X FROM WHICH TO BRANCH

C  MODIFICATION TO ORIGINAL ALGORITHM FOR DETERMINING BRANCH
C  POINT - - BRANCH TO THE RIGHT WHENEVER A TOUR IS NOT
C  COMPLETED OR BRANCH FROM THE LOWEST NUMBERED YBAR NODE
C  WHICH IS A TERMINAL NODE, IN THE ORDER GIVEN.

  799 YTAB(ITER,7) = 0
```

```
C  BRANCH TO RIGHT IS EXHAUSTED, THEREFOR SEARCH YBAR NODES
C  FOR A FEASIBLE LABEL (I.E. LABEL.LE.ZO)

  800 DO 830 I=LABEL,ITER
      IF(YBTAB(I,4).EQ.0) GO TO 830
      IF(YBTAB(I,3).GT.ZO) YBTAB(I,4) = 0
      IF(YBTAB(I,3).GT.ZO) GO TO 830
      LABEL = I
      INDEX = YBTAB(I,1)
      CALL SETUP (&500)
  830 CONTINUE
C  OPTIMAL ROUTE HAS BEEN FOUND : TERMINATE

      CALL TIMEIT (-1,TIMEX)
      RETURN

C  ERROR MESSAGES HAVE BEEN PRODUCED: TERMINATE CASE

  840 WRITE(6,15)
      RETURN

C  BRANCH TO RIGHT IS NOT EXHAUSTED: THEREFOR BRANCH FROM
C  Y NODE AND MAKE NODE.Y NON-TERMINAL


  850 YTAB(ITER,7) = 0
      INDEX = YTAB(ITER,1)
      WX = WY
      GO TO 500
      END
```

```
      SUBRCUTINE SETUP (*)

      IMPLICIT INTEGER*2(A-Z)
      REAL*4 TIMEX
      INTEGER*4 M(20,20,19),THETA,MAXEL,MINEL(20),MIN(20),
     1TITLE(17),ZC,WY
      DIMENSION A(20,20,19),COST(20),LEGCOM(20),TCOST(30)
     1FM(20),TO(20),X(20),ARCCOM(20,20),BEST(20,4)
     2YTAB(2500,7),YBTAB(2500,2),IK(20),JK(20)
      COMMCN TIMEX,M,THETA,MAXEL,MINEL,MIN,TITLE,ZO,WY,A,
     1COST,LEGCOM,FM,TO,X,ARCCOM,BEST,YTAB,YBTAB,IK,JK,TOUR,
     2AA,N,L,ALIKE,LEGREQ,STEP,ITER,DEL,WX,MAXLEG,LEG,WYBAR,
     3IO,JO,KO,ITS,TCOST,INDEX
C   STEP NINE
      LEG = 0
      G = 0
      NCCM = 0

      IF(INDEX.LT.0) YBTAB(-INDEX,2) = 0
      IF(INDEX.GT.0) YTAB(INDEX,7) = 0
      DO 901 I=1,N
      LEGCOM(I) = 0
      FM(I) = 0
      TO(I) = 0
  901 COST(I) = 0
      DO 902 I=1,N
      DO 902 J=1,N
  902 ARCCCM(I,J) = 100
C   STEP 9  SUBSTEP 1
C   COMPUTE G=SUM A(I,J,K) FOR COMMITTED ARCS AND LEGS
      BB = INDEX
      IF(INDEX.LT.0) FROM = YTAB(-INDEX,2)
      IF(INDEX.GT.0) FROM = YTAB(INDEX,2)
      DO 909 I=1,ITER
      IF (BB.GT.0) GO TO 903
      IF (FROM.EQ.-1.OR.FROM.EQ.0) GO TO 910
      IF (FROM.LT.0) GO TO 908
      GO TO 904
C   LSED TO START EACK TREE FROM BRANCH NODE AND
C   CONSIDER THAT NODE
  903 FROM = BB

  904 G = G + A(YTAB(FROM,4),YTAB(FROM,5),YTAB(FROM,6))
      LEGCOM(YTAB(FROM,6)) = YTAB(FROM,6)
      KO = YTAB(FROM,6)
      FM(YTAB(FROM,6)) = YTAB(FROM,4)
      IO = YTAB(FROM,4)
      TC(YTAB(FROM,6)) = YTAB(FROM,5)
      JO = YTAB(FROM,5)
      COST(YTAB(FROM,6)) = A(IO,JO,KO)
      DO 905 AI=1,N
      DO 905 BJ=1,N
      IF((AI.EQ.IC.OR.AI.EQ.JO.OR.BJ.EQ.IO.OR.BJ.EQ.JO).AND.
     1ARCCOM(AI,BJ).GT.99) ARCCOM(AI,BJ) = -99
  905 CONTINUE
      NCOM = NCOM + 1
      FROM = YTAB(FROM,2)
      BB =-1000
      GO TO 909

  908 FROM = YTAB(-FROM,2)
  909 CONTINUE
```

```
C   STEP 9   SUBSTEP 2   SETTING UP M(K)

  910 LEG = NCCM
      DO 911 K=1,L
      DO 911 I=1,N
      DO 911 J=1,N
  911 M(I,J,K) = A(I,J,K)

C   STEP 9   SUBSTEP 3
C   DELETE ARCS AND LEGS COMMITTED

      GOLF = 1
  912 DEL = GOLF
      DO 913 F=1,L
      IF (FM(F).EQ.DEL) GOLF = GOLF + 1
      IF (FM(F).EQ.DEL) GO TO 912
      IF (TO(F).EQ.DEL) GOLF = GOLF + 1
      IF (TO(F).EQ.DEL) GO TO 912
  913 CONTINUE

C   STEP 9   SUBSTEP 4   BLOCK PATHS NOT ALLOWED

      IF(INDEX.EQ.-1)M(YTAB(1,4),YTAB(1,5),YTAB(1,6))=32000
      IF(INDEX.EQ.-1) GO TO 919
      IF (INDEX.LT.0) M(YTAB(-INDEX,4),YTAB(-INDEX,5),YTAB(
     1-INDEX,6)) = 32000
      IF(INDEX.LT.0) FROM = YTAB(-INDEX,2)
      IF(INDEX.GT.0) FROM = YTAB(INDEX,2)
      DO 918 I=1,ITER
      IF (FROM.EQ.0) GO TO 919
      IF (FROM.EQ.-1) GO TO 916
      IF (FROM.LT.0) GO TO 917
      FROM = YTAB(FROM,2)
      GO TO 918
  917 M(YTAB(-FROM,4),YTAB(-FROM,5),YTAB(-FROM,6)) = 32000
      FROM = YTAB(-FROM,2)
  918 CONTINUE
  916 M(YTAB(1,4),YTAB(1,5),YTAB(1,6)) = 32000

C   STEP 9 SUBSTEP 5   FIND MINIMUM ELEMENT IN EACH MATRIX


  919 CALL MINELM


      WX = G
      DO 940 K=1,L
      IF (K.EQ.LEGCOM(K)) GO TO 940
      WX =WX + MINEL(K)
  940 CONTINUE
      RETURN 1
      END
```

```
      SUBROUTINE ROUTE (*,*)

      IMPLICIT INTEGER*2(A-Z)
      REAL*4 TIMEX
      INTEGER*4 M(20,20,19),THETA,MAXEL,MINEL(20),MIN(20),
     1TITLE(17),ZO,WY
      DIMENSION A(20,20,19),COST(20),LEGCOM(20),TCOST(30)
     1FM(20),TO(20),X(20),ARCCOM(20,20),BEST(20,4)
     2YTAB(2500,7),YBTAB(2500,2),IK(20),JK(20)
      COMMON TIMEX,M,THETA,MAXEL,MINEL,MIN,TITLE,ZO,WY,A,
     1COST,LEGCOM,FM,TO,X,ARCCOM,BEST,YTAB,YBTAB,IK,JK,TOUR,
     2AA,N,L,ALIKE,LEGREQ,STEP,ITER,DEL,WX,MAXLEG,LEG,WYBAR,
     3IO,JO,KO,ITS,TCOST,INDEX

C   STEP 10

   20 FORMAT('0',10X,'THE TOUR NUMBER IS EQUAL TO 31, AND',
     1' THE VARIABLE TCOST IS ONLY DIMENSIONED FOR 30 TOURS'
     2,':  CASE TERMINATED.')
   21 FORMAT('0',5X,'STEP NO.',I3,' ITER NO.',I6,': UPPER',
     1' BOUND ON VALUE OF OPTIMAL TOUR IS',I5)
   23 FORMAT ('0',5X,'FEASIBLE TOUR NO.',I3,' IS AS FOLLOWS:


      ZO = WY
      WRITE(6,21) STEP,ITER,ZO

C   NODE IS MADE NON-TERMINAL SINCE A ROUTE HAS BEEN
C   COMPLETED AND NO BRANCHING CAN TAKE PLACE.

      YTAB(ITER,7) = 0
      TOUR = TOUR + 1

C   ERROR MESSAGE: THE NUMBER OF TOURS IS.GT.TCOST DIMENSION

      IF(TOUR.GT.30) WRITE(6,20)
      IF(TOUR.GT.30) RETURN 2

      DO 985 I=1,N
  985 X(I) = 0

C   DETERMINE BY PROCESS OF ELIMINATION AND ORDERING
C   WHAT LEG OF ROUTE IS MISSING AND THUS FORM COMPLETED
C   ROUTE.

 1000 DO 1020 K=1,L
      IF (K.EQ.LEGCOM(K)) GO TO 1020
      LEGCOM(K) = K
      IF (K.NE.1) GO TO 1010
      TO(1) = FM(2)
      DO 1002 I=1,L
      IF (FM(I).NE.0) X(FM(I)) = FM(I)
 1002 CONTINUE
      X(TO(L)) = TO(L)
      DO 1003 I=1,N
      IF (X(I).NE.0) GO TO 1003
      FM(1) = I
      COST(1) = A(FM(1),TO(1),1)
      GO TO 1021
 1003 CONTINUE
      GO TO 1020

 1010 IF (K.EQ.L) GO TO 1011
      FM(K) = TO(K-1)
      TO(K) = FM(K+1)
      COST(K) = A(FM(K),TO(K),K)
      GO TO 1021
 1011 FM(L) = TO(L-1)
      DO 1012 I=1,L
      IF (TO(I).NE.0) X(TO(I)) = TO(I)
 1012 CONTINUE
```

```
      X(FM(1)) = FM(1)
      DO 1013 I=1,N
      IF (X(I).NE.0) GO TO 1013
      TO(L) = I
      COST(L) = A(FM(L),TO(L),L)
      GO TO 1021
 1013 CONTINUE
 1020 CONTINUE


 1021 TCOST(TOUR) = 0
      DO 1030 K=1,L
 1030 TCOST(TOUR) = TCOST(TOUR) + COST(K)

C  COMPLETE TOUR IS NOW KNOWN.  ENTER IT IN MATRIX BEST.

      IF(TOUR.EQ.1) GO TO 1040
      IF(TCOST(TOUR).GE.BEST(N,2)) GO TO 1060
 1040 DO 1050 K=1,L
      BEST (K,1) = K
      BEST (K,2) = FM(K)
      BEST (K,3) = TO(K)
 1050 BEST (K,4) = COST(K)
      BEST(N,1) = TOUR
      BEST(N,2) = TCOST(TOUR)
 1060 RETURN 1
      END
```

```
      SUBROUTINE SOLN

      IMPLICIT INTEGER*2(A-Z)
      REAL*4 TIMEX
      INTEGER*4 M(20,20,19),THETA,MAXEL,MINEL(20),MIN(20),
     1TITLE(17),ZC,WY
      DIMENSION A(20,20,19),COST(20),LEGCOM(20),TCOST(30)
     1FM(20),TO(20),X(20),ARCCOM(20,20),BEST(20,4)
     2YTAB(2500,7),YBTAB(2500,2),IK(20),JK(20)
      COMMCN TIMEX,M,THETA,MAXEL,MINEL,MIN,TITLE,ZO,WY,A,
     1COST,LEGCOM,FM,TO,X,ARCCOM,BEST,YTAB,YBTAB,IK,JK,TOUR,
     2AA,N,L,ALIKE,LEGREQ,STEP,ITER,DEL,WX,MAXLEG,LEG,WYBAR,
     3IO,JO,KO,ITS,TCOST,INDEX

    4 FORMAT(1H1)
    5 FORMAT('0',///,25X,'ITERATION INFORMATION')
   15 FORMAT('0',10X 'NUMBER OF ALTERNATE OPTIMAL TOURS =',
     1 I4)
   16 FORMAT('0',10X,'BEST TOUR SO FAR IS AS FOLLOWS:')
   17 FORMAT('0',26X,'ROUTE COST = ',I5)
   18 FORMAT('0',10X,'FEASIBLE TOUR NO.',I3,' IS DECLARED',
     1' OPTIMAL')
   19 FORMAT ('0',10X,'************************************')
   24 FORMAT(' ',7X,3I8,4I5,4X,2I8)
   27 FORMAT('0',10X,'LEG',5X,'FROM',5X,' TO',5X,'COST')
   28 FORMAT('0',11X,I2,7X,I2,6X,I2,5X,I4)
   31 FORMAT('0',27X,'YTABLE',27X,'YBARTABLE',//,12X,
     1'NODE     FRCM       WY    IO    JO    KO TERM',7X,
     2'WYBAR      TERM')
   48 FORMAT('0',10X,'NUMBER OF ITERATIONS REQUIRED =',I7)
   50 FORMAT('0',13X,'OPTIMAL ROUTE COST = ',I5)
   55 FORMAT('0',10X,'TIME TO COMPUTE SOLUTION=',-6PF15.6,
     1' SECONDS')
   56 FORMAT('0',10X,'TIME USED UP SO FAR = ',-6PF15.6,
     1' SECONDS')


      IF(ALIKE.EQ.1) BEST(L,3) = 1
      IF(ITER.GT.ITS) GO TO 1520
      WRITE(6,19)
      WRITE(6,18) BEST(N,1)
 1520 IF(ITER.GT.ITS) WRITE(6,16)
      WRITE(6,27)
      DO 1550 K=1,L
 1550 WRITE(6,28) (BEST(K,J),J=1,4)
      IF(ITER.LE.ITS) WRITE(6,50) BEST(N,2)
      IF(ITER.GT.ITS) WRITE(6,17) BEST(N,2)
      WRITE(6,19)
      IF(ITER.GT.ITS) GO TO 1575
      ALT = -1
      DO 1530 I=1,TOUR
      IF(TCOST(I).GT.BEST(N,2)) GO TO 1530
      ALT = ALT + 1
 1530 CONTINUE
      WRITE(6,15) ALT
      WRITE(6,48) ITER

 1575 IF(ITER.LE.ITS) WRITE(6,55) TIMEX
      IF(ITER.GT.ITS) WRITE(6,56) TIMEX

C  WRITE OUT ITERATION INFORMATION

      WRITE(6,4)
      WRITE(6,5)
      IF(ITER.GT.ITS) ITER = ITER - 1
      WRITE(6,31)

      DO 1600 I=1,ITER
 1600 WRITE(6,24) (YTAB(I,J),J=1,7),(YBTAB(I,J),J=1,2)
      RETURN
      END
```

```
      SUBROUTINE MINELM

      IMPLICIT INTEGER*2(A-Z)
      REAL*4 TIMEX
      INTEGER*4 M(20,20,19),THETA,MAXEL,MINEL(20),MIN(20),
     1TITLE(17),ZO,WY
      DIMENSION A(20,20,19),COST(20),LEGCOM(20),TCOST(30)
     1FM(20),TO(20),X(20),ARCCOM(20,20),BEST(20,4)
     2YTAB(2500,7),YBTAB(2500,2),IK(20),JK(20)
      COMMON TIMEX,M,THETA,MAXEL,MINEL,MIN,TITLE,ZO,WY,A,
     1COST,LEGCOM,FM,TO,X,ARCCOM,BEST,YTAB,YBTAB,IK,JK,TOUR,
     2AA,N,L,ALIKE,LEGREQ,STEP,ITER,DEL,WX,MAXLEG,LEG,WYBAR,
     3IO,JO,KO,ITS,TCOST,INDEX

      DO 630 K=1,L
      KEY = 0
      MINEL(K) = -1
      IK(K) = 0
      JK(K) = 0
      IF (K.EQ.LEGCOM(K)) GO TO 630
  602 MIN(K) = 32000
      DO 629 I=1,N
      DO 629 J=1,N
      IF(I.EQ.J) GO TO 621
      IF (K.EQ.1.AND.LEGCOM(2).EQ.0) GO TO 605
      IF (K.EQ.1) GO TO 606
      IF (K.EQ.L.AND.LEGCOM(L-1).EQ.0) GO TO 605
      IF (K.EQ.L) GO TO 610
      IF(K.EQ.LEGCOM(K+1)-1.AND.K.NE.LEGCOM(K-1)+1)GOTO 606
      IF(K.EQ.LEGCOM(K-1)+1.AND.K.NE.LEGCOM(K+1)-1)GOTO 610
      IF(K.EQ.LEGCOM(K+1)-1.AND.K.EQ.LEGCOM(K-1)+1)GOTO 609
  605 IF (ARCCOM(I,J).LT.100) GO TO 621
      IF (KEY.EQ.0) GO TO 620
      IF (KEY.EQ.1) GO TO 622
  606 IF (J.NE.FM(K+1)) GO TO 621
      IF (ARCCOM(I,DEL).LT.100) GO TO 621
      IF (KEY.EQ.0) GO TO 620
      IF (KEY.EQ.1) GO TO 622
  609 IF(I.EQ.TO(K-1).AND.J.EQ.FM(K+1).AND.KEY.EQ.0)GOTO620
      IF(I.EQ.TO(K-1).AND.J.EQ.FM(K+1).AND.KEY.EQ.1)GOTO622
      GO TO 621
  610 IF (I.NE.TO(K-1)) GO TO 621
      IF (ARCCOM(DEL,J).LT.100) GO TO 621
      IF (KEY.EQ.1) GO TO 622
  620 MINEL(K) = MINO(MIN(K),M(I,J,K))
      IF (MINEL(K).GE.MIN(K)) GO TO 621
      IK(K) = I
      JK(K) = J
      MIN(K) = MINEL(K)
  621 IF (KEY.EQ.1) GO TO 629
      IF (I.NE.N) GO TO 629
      IF (J.NE.N) GO TO 629

C   REDUCE MATRICES

      KEY = 1
      GO TO 602
  622 IF(I.EQ.J) GO TO 629
      M(I,J,K) = M(I,J,K) - MINEL(K)
  629 CONTINUE
  630 CONTINUE

      RETURN
      END
```

```
      SUBROUTINE CHECK

C  THIS SUBROUTINE DETERMINES IF A COMPLETE ROUTE CAN BE
C  ENUMERATED AFTER ONLY EVERY OTHER LEG HAS BEEN
C  DETERMINED, STARTING WITH LEG ONE

      IMPLICIT INTEGER*2(A-Z)
      REAL*4 TIMEX
      INTEGER*4 M(20,20,19),THETA,MAXEL,MINEL(20),MIN(20),
     1TITLE(17),ZO,WY
      DIMENSION A(20,20,19),COST(20),LEGCOM(20),TCOST(30)
     1FM(20),TO(20),X(20),ARCCOM(20,20),BEST(20,4)
     2YTAB(2500,7),YBTAB(2500,2),IK(20),JK(20)
      COMMON TIMEX,M,THETA,MAXEL,MINEL,MIN,TITLE,ZO,WY,A,
     1COST,LEGCOM,FM,TO,X,ARCCOM,BEST,YTAB,YBTAB,IK,JK,TOUR,
     2AA,N,L,ALIKE,LEGREQ,STEP,ITER,DEL,WX,MAXLEG,LEG,WYBAR,
     3IO,JO,KO,ITS,TCOST,INDEX

      DO 200 K=1,L,2
      IF (LEGCOM(K).NE.0) GO TO 200
      RETURN 1
  200 CONTINUE
      LAST = N-4
      DO 300 K=2,LAST,2
      LEGCOM(K) = K
      FM(K) = TO(K-1)
      TO(K) = FM(K+1)
  300 COST(K) = A(FM(K),TO(K),K)
      RETURN
      END
```

```
      SUBROUTINE TIMEIT

C  NN=0 STARTS CLOCK:  NN=-1 STOPS CLOCK

      IT = NN+2
      GO TO (20,10),IT
   10 CALL TIMON(MM)
      TIMEM = MM
      RETURN
   20 CALL TIMOFF(MM)
      TIME = MM
      TIME=(TIMEM-TIME) * 26.0
      RETURN
      END


C  ASSEMBLY LANGUAGE LISTING TO CONDUCT TIMING ROUTINE

TIMEALL  CSECT
         ENTRY   TIMON,TIMOFF
TIMON    SAVE    (14,12)             ENTRY VIA -CALL TIMON(N)-
         USING   TIMON,12
         LR      12,15
         ST      13,TEMP1
         LA      13,SAVE1
         L       2,0(1,0)
         L       3,TOTIME
         ST      3,CLOCKR
         ST      3,0(2,0)
         STIMER  TASK,TUINTVL=CLOCKR
         L       13,TEMP1
EXIT     RETURN  (14,12),T,RC=0
TIMOFF   SAVE    (14,12)             ENTER VIA - CALL TIMOFF(N)
         USING   TIMOFF,12
         LR      12,15
         ST      13,TEMP1
         LA      13,SAVE1
         L       2,0(1,0)
         TTIMER  CANCEL
         ST      0,0(2,0)
         L       13,TEMP1
         RETURN  (14,12),T,RC=0
         CNOP    0,4
TOTIME   DC      X'7FFFFFFF'
CLOCKR   DS      F
SAVE1    DS      18F
TEMP1    DS      F
         END
```

APPENDIX F

MODIFICATIONS TO COMPUTER PROGRAM FOR DYNAMIC STORAGE ALLOCATION

The following modifications to the basic program presented in Appendix E will provide dynamic storage allocation based on the number of nodes (N) and the maximum number of iterations (ITS) desired for each case in the computer data deck:

1. The Assembly Language listing on the following pages should be inserted at the very front of the computer source deck.

2. A new main program which is found after the Assembly Language listing replaces the original main program. The original main program becomes SUBROUTINE START and is listed here after the new main program.

3. All other subroutines remain the same as before with the exception of the variable type specification statements, DIMENSION statements, and COMMON statements. These 6 statements as found in the new SUBROUTINE START must be used in all the old FORTRAN subroutines except SUBROUTINE TIMEIT which does not change.

4. The CALLS for the subroutines and the SUBROUTINE definition cards must be the same as before with the added arguments as found in the new SUBROUTINE START definition card.

5. The JCL is included as a guide and is unique to the IBM 360 Model 67 Computer System installation at the Naval Postgraduate School. The only card which is required to be changed on various

runs in the EXEC card. It must contain a region for the GO step which is large enough to handle the case with the maximum number of iterations and specify time for the GO step large enough to accommodate the expected running time for all cases in the data deck.

The makeup of the revised computer deck is on the following page.

ORGANIZATICN OF COMPUTER CARD DECK USING REVISED PROGRAM

```
//    (THIS CARD SHOULD CONTAIN JOB NAME AND OTHER PERTINENT INFORMATION)
//ASM1 EXEC   PGM=IEUASM,REGION=58K,PARM='LOAD,NODECK,LINECNT=75'
//SYSLIB DD     DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1 DD     UNIT=SYSDA,SPACE=(CYL,(5,5))
//SYSUT2 DD     UNIT=SYSDA,SPACE=(CYL,(5,5))
//SYSUT3 DD     UNIT=(SYSDA,SEP=(SYSUT2,SYSUT1,SYSLIB)),SPACE=(CYL,(5,5))
//SYSPRINT DD    SYSOUT=A,DCB=(RECFM=FBM,LRECL=121,BLKSIZE=3388)
//SYSGO  DD     UNIT=SYSDA,DISP=(NEW,PASS),DSNAME=&SYSLIN,              *
//              SPACE=(CYL,(3,2)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//ASM1.SYSIN DD *

( INSERT ASSEMBLER SOURCE DECK FOR DYNAMIC STORAGE ALLOCATION )

/*
//S2 EXEC FRTHCALG,REGION.GO=156K,TIME.GO=15
//FORT.SYSLIN DD DISP=(MOD,PASS)
//FORT.SYSIN  DD *

( INSERT NEW MAIN PROGRAM AND THEN ALL FORTRAN SUBROUTINES INCLUDING
  SUBROUTINE START )

/*
//ASM.SYSIN DD *

( INSERT ASSENBLER SOURCE DECK FOR TIMING ROUTINE )

/*
//GO.SYSIN DD *

( INSERT DATA DECK )

/*
```

```
C   ASSEMBLY LANGUAGE PROGRAM USED TO OBTAIN DYNAMIC
C   ALLOCATION OF STORAGE SPACE BASED ON THE INPUT PARAMETERS
C   N AND ITS

         MACRO
         REGS
         LCLA   &N
         LCLC   &SYM
.LOOP    ANOP
&SYM     SETC   '&N'
R&SYM    EQU    &SYM
&N       SETA   &N+1
         AIF    (&N LT 16).LOOP
         MEND
GETARY   CSECT
         REGS
         STM    R14,R12,12(R13)       SAVE REGS
         LR     R12,R15               GET BASE OUT OF R15
         USING  GETARY,R12
         LA     R11,SAVEAREA
         ST     R13,4(R11)            LINK SAVE AREAS
         ST     R11,8(R13)
         LR     R13,R11
         LR     R11,R1  SAVE ARGUMENT LIST LOC FROM GETMAIN
         USING  ARGS,R11
         L      R2,ANUM               GET NUM ARGUMENT
         L      R3,0(R2)              CHECK ITS VALIDITY
         LTR    R3,R3
         BC     13,NUMERR1            NUM NEGATIVE OR ZERO
         LA     R4,99
         CR     R3,R4
         BH     NUMERR2               NUM GT 99
         LR     R4,R3          LENGTH OF SCRATCH = 8*NUM + 8
         SLL    R4,3
         LA     R0,8(R4)
         GETMAIN R,LV=(0)
         LR     R9,R1
         USING  SCRATCH,R9
         ST     R3,NUM START UNPACKING AND CHECKING CALL LIST
         L      R5,ANEXT
         L      R5,0(R5)
         CL     R5,=X'000C0000' ADDRESS NEGATIVE OR GT 768K ?
         BH     BADNEXT
         ST     R5,NEXT
         SRL    R4,1      ISSUE GETMAIN FOR NEXT ARGUMENT LIST
         LR     R0,R4
         GETMAIN R,LV=(0)
         LR     R10,R1
         USING  CALLLIST,R10
         LA     R7,AL1
         LA     R6,L1
         LA     R4,ARRY1
GETCORE  L      R5,0(R7)         GET ADDRESS OF HUNK LENGTH
         LTR    R5,R5            LAST ONE ?
         BP     CONTIN           NOPE
         CH     R3,=H'1'  DOES NUM AGREE ON LAST ONE?
         BE     CONTIN1   YES
         B      NUMERR3   NO
CONTIN   CH     R3,=H'1' DOES NUM SAY IT IS LAST AND IT ISNT?
         BE     NUMERR4
CONTIN1  L      R5,0(R5)  GET LNGTH
         CL     R5,=X'00080000' LENGTH GT 512K OR NEGATIVE?
         BH     BADLENG
         LR     R0,R5
         GETMAIN R,LV=(0)
         ST     R1,0(R4)         PUT ADDRESS IN CALL LIST
         LA     R4,4(R4)         INCREMENT FOR NEXT LOOP
         ST     R5,0(R6) PUT LTH IN SCRATCH FOR LATER FREEMAN
         ST     R1,4(R6)         PUT ADDRESS IN SCRATCH
         LA     R6,8(R6)         INCREMENT
         LA     R7,4(R7) HUNK LENGTH AND SCRTCH AREA REGS
```

```
         BCT    R3,GETCORE       GET NEXT ARRAY
         SH     R4,=H'4'         PUT HEX 80 ON LAST ADDRESS
         MVI    0(R4),X'80'
         LR     R1,R10          PUT ADDRESS OR CALL LIST IN R1
         L      R15,NEXT
         BALR   R14,R15         CALL NEXT ROUTINE
         L      R0,NUM          GET RID OF CALL LIST
         SLL    R0,2
         LA     R1,ARRY1
         FREEMAIN R,LV=(0),A=(1)
         LA     R5,L1           LOOP TO FREE ARRAY CORE
         L      R6,NUM
FREECOR  L      R0,0(R5)        GET LENGTH OF FIRST ARRAY HUNK
         L      R1,4(R5)           GET ITS ADDRESS
         FREEMAIN R,LV=(0),A=(1)
         LA     R5,8(R5) INCREMENT POINT TO HUNK LOC AND LGTH
         BCT    R6,FREECOR
         L      R0,NUM          FREE SCRATCH AREA
         SLL    R0,3
         LA     R0,8(R0)        NUM*8 +8 IN R0
         LA     R1,SCRATCH
         FREEMAIN R,LV=(0),A=(1)
         L      R7,AERRMSG
         MVI    0(R7),X'40'BLANK OUT ERROR MSG(NORMAL RETURN)
         MVC    1(31,R7),0(R7)
RETURN   L      R13,4(R13) ALL CLEANED UP, RETURN TO CALLING
         LM     R14,R12,12(R13)        PROGRAM
         SR     R15,R15
         BR     R14
NUMERR1  LA     R6,MSG1         INSERT APPROPRIATE ERROR MSG
         B      INSERT             AND RETURN DIRECTLY TO CALLI
NUMERR2  LA     R6,MSG2            PROGRAM
         B      INSERT
NUMERR3  LA     R6,MSG3
         B      INSERT
NUMERR4  LA     R6,MSG4
         B      INSERT
BADNEXT  LA     R6,MSG5
         B      INSERT
BADLENG  LA     R6,MSG6
         L      R4,NUM
         LA     R4,1(R4)
         SR     R4,R3     NUMBER OF BAD ARGUMENT IN R4
         CVD    R4,MSG1 PUT IT IN CHARACTER FORM AND CLOBBER
         UNPK   MSG1+8(3),MSG1+6(2)       MSG1
         OI     MSG1+10,X'F0'   CHANGE BOTTOM ZONE TO FOX
         MVC    MSG6+7(2),MSG1+9    PUT IT IN TEXT OF MSG6
INSERT   L      R7,AERRMSG
         MVC    0(32,R7),0(R6)
         B      RETURN
         DS     0F
SAVEAREA DC     18F'0'
         DS     0D
MSG1     DC     C' NUM OF ARRAYS - OR 0            '
MSG2     DC     C' NUM OF ARRAYS > 99              '
MSG3     DC     C' NUM OF ARRAYS > NUM GIVEN LNGTS'
MSG4     DC     C' NUM OF ARRAYS < NUM GIVEN LNGTS'
MSG5     DC     C' ADDR NEXT ROUTINE - OR > 768K  '
MSG6     DC     C' LENGTH      IS - OR > 512K      '
ARGS     DSECT
AERRMSG  DC     A(0)   ADD OF 32 BYTES FOR ERROR TEXT
ANEXT    DC     A(0)   ADD OF NEXT ROUTINE TO BE CALLED
ANUM     DC     A(0)   ADD OF NUMBER OF HUNKS OF CORE WANTED
AL1      DC     A(0)   ADD OF 1ST HUNK BYTE LENGTH
ALNUM    DC     X'80',AL3(0)
SCRATCH  DSECT
NEXT     DC     A(0)
NUM      DC     A(0)
L1       DC     F'0'
A1       DC     A(0)
LLAST    DC     F'0'
ALAST    DC     A(0)
CALLLIST DSECT
ARRY1    DC     A(0)
ARRYNUM  DC     A(0)
         END
```

```
C  NEW MAIN PROGRAM FOR DYNAMIC ALLOCATION OF STORAGE SPACE

      EXTERNAL START
      INTEGER*2 ALIKE,AA,NCASE
      INTEGER*4 ERROR(8),BLNK/4H    /
      DIMENSION TITLE(17)
      COMMON/Z/ TITLE,N,ITS,AA,ALIKE


    1 FORMAT(I4)
    2 FORMAT(I4,I2,I6,17A4)
   25 FORMAT(8A4)

      READ(5,1) NCASE

      DO 10 AA=1,NCASE

C  READ INPUT PARAMETERS

      READ(5,2) N,ALIKE,ITS,(TITLE(I),I=1,17)

C  CALCULATE AMOUNT OF STORAGE REQUIRED FOR VARIOUS ARRAYS

      NNNM12 = N*N*(N-1)*2
      NNNM14=NNNM12*2
      N15 = 15
      N2 = N*2
      N4 = N*4
      NN2 = N*N2
      N42 = N4*2
      ITS72 = ITS*14
      ITS22 = ITS*4

C  CALL ASSEMBLY LANGUAGE PROGRAM FOR OBTAINING STORAGE

      CALL GETARY(ERROR,START,N15,NNNM14,N4,N4,NNNM12,N2,N2,
     1N2,N2,N2,NN2,N42,ITS72,ITS22,N2,N2)

      IF(ERROR(1).NE.BLNK) GO TO 20
   10 CONTINUE

      GO TO 30
   20 WRITE(6,25) ERROR
   30 STOP
      END
```

```
C  THIS SUBROUTINE, START, IS THE MODIFIED OLD MAIN PROGRAM

C  THE FIRST 11 CARDS OF THIS PROGRAM MUST BE USED IN ALL
C  OTHER FORTRAN SUBROUTINES EXCEPT SUBROUTINE TIMEIT, AND
C  THE CALLS AND OTHER SUBROUTINE CARDS MUST HAVE THE SAME
C  ARGUMENTS AS THOSE IN THE CALLS IN THIS SUBROUTINE, WITH
C  THE EXCEPTION OF THOSE WITH SPECIAL STATEMENT NUMBERS AS
C  ARGUMENTS IN THE OLD PROGRAM, AND THESE MUST APPEAR AT
C  THE HEAD OF THE ARGUMENT LIST.

      SUBROUTINE START (M,MINEL,MIN,A,COST,LEGCOM,FM,TO,X,
     1ARCCOM,BEST,YTAB,YBTAB,IK,JK)


      IMPLICIT INTEGER*2(A-Z)
      REAL*4 TIMEX
      INTEGER*4 N,ITS
      INTEGER*4 THETA,MAXEL,TITLE(17),ZO,WY
      INTEGER*4 M(N,N,N),MINEL(N),MIN(N)
      DIMENSION A(N,N,N),COST(N),LEGCOM(N),TCOST(30),FM(N),
     1TO(N),X(N),ARCCOM(N,N),BEST(N,4),IK(N),JK(N),
     2YTAB(ITS,7),YBTAB(ITS,4)
      COMMON TIMEX,THETA,MAXEL,ZO,WY,TOUR,L,LEGREQ,STEP,DEL,
     1ITER,WX,MAXLEG,LEG,WYBAR,IO,JO,KO,TCOST,INDEX
      COMMON/Z/ TITLE,N,ITS,AA,ALIKE


      CALL INPUT (M,MINEL,MIN,A,COST,LEGCOM,FM,TO,X,
     1ARCCOM,BEST,YTAB,YBTAB,IK,JK)

      CALL ITERTE (&2000,M,MINEL,MIN,A,COST,LEGCOM,FM,TO,X,
     1ARCCOM,BEST,YTAB,YBTAB,IK,JK)

      CALL SOLN (M,MINEL,MIN,A,COST,LEGCOM,FM,TO,X,
     1ARCCOM,BEST,YTAB,YBTAB,IK,JK)

 2000 CONTINUE
      RETURN
      END
```

BIBLIOGRAPHY

1. DeHaemer, M. J., A Branch-and-Bound Algorithm for the Solution of Sequence-Dependent Routing Problems, M.S. Thesis, U. S. Naval Postgraduate School, Monterey, California, April 1970.

2. Sweeney, Dura W., The Exploration of a New Algorithm for Solving the Traveling-Salesman Problem, M.S. Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1963.

3. Lawler, E. L., and Wood, E.D., "Branch-and-Bound Methods: A Survey," Operations Research, v. 14, p. 699-719, 1966.

4. Arick, John C., A Computer Program for Integer Solutions to Linear Programming Problems, M.S. Thesis, U. S. Naval Postgraduate School, Monterey, California, October 1969.

5. Little, J.D.C., and others, "An Algorithm for the Traveling Salesman Problem," Operations Research, v. 11, p. 972-989, 1963.

6. Dantzig, G.B., Fulkerson, D.R., and Johnson, S.M., "Solutions of a Large-Scale Traveling-Salesman Problem," Operations Research, v. 2 , p. 393-410, 1954.

7. Dantzig, G.B., Fulkerson, D.R., and Johnson, S.M., "On a Linear-Programming, Combinatorial Approach to the Traveling-Salesman Problem," Operations Research, v. 7, p. 58-63, 1959.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Professor R. H. Shudde, Code 55Su Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	1
4.	Department of Operations Analysis, Code 55 Naval Postgraduate School Monterey, California 93940	1
5.	LT Richard A. Jackson, USN U. S. Naval Destroyer School Class 34 Newport, Rhode Island 02840	1
6.	LCDR Michael J. DeHaemer, USN USS Whale (SSN-638) FPO New York 09501	1
7.	Special Projects Office Department of the Navy (SSP-114) 1105 Crystal Mall Jerrerson Davis Highway Arlington, Virginia 20390 Attn: CDR Richard Franzen, USN	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

A Computer Program For Solution of Sequence Dependent Routing Problems Using a Branch-And-Bound Algorithm

4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)

Master's Thesis; September 1970

5. AUTHOR(S) (First name, middle initial, last name)

Richard Alan Jackson

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
September 1970	70	7

8a. CONTRACT OR GRANT NO. b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

An algorithm for the solution of sequence-dependent routing problems is presented and programmed in FORTRAN IV for use on digital computers. Solutions, computation times and iteration requirements are summarized and discussed for eleven test cases.

With specific modification of the input data, a typical traveling salesman closed-loop problem may be solved by the same program.

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Branch-and-Bound						
Sequence-Dependent Routing						
Traveling Salesman						
Routing						

DD FORM 1 NOV 65 1473 (BACK)

S/N 0101-807-6821

Security Classification

A-31409

27 MAY 71 19796
18 APR 72 19641
30 MAY 75 23333
26 JUL 76 23817

122753

Thesis
J225 Jackson
c.1 A computer program for solution of sequence dependent routing problems using a branch-and-bound algorithm.